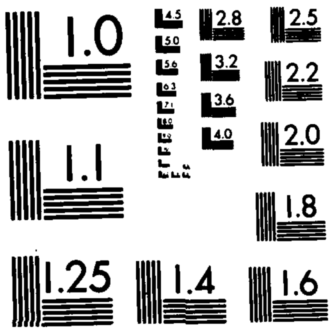
AD-A138 892 DISTRIBUTED DATABASE CONTROL AND ALLOCATION VOLUME 2 1/2
PERFORMANCE ANALYSIS..(U) COMPUTER CORP OF AMERICA
CAMBRIDGE MA W K LIN ET AL. OCT 83
UNCLASSIFIED RADC-TR-83-226-VOL-2 F30602-81-C-0028 F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(12)

RADC-TR-83-226, Vol II (of three)
Final Technical Report
October 1983

DISTRIBUTED DATABASE CONTROL AND ALLOCATION Performance Analysis of Concurrency Control Algorithms

Computer Corporation of America

Wente K. Lin, Philip A. Bernstein, Nathan Goodman and Jerry Nolte

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

RADC-TR-83-226, Vol II (of three) has been reviewed and is approved for publication.

APPROVED: Emilie J. Siarkiewicz

EMILIE J. SIARKIEWICZ
Project Engineer

APPROVED: [illegible]

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: [illegible]

DONALD A. BRANTINGHAM
Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-83-226, Vol II (of three)	2. GOVT ACCESSION NO. AD-A138892	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DISTRIBUTED DATABASE CONTROL AND ALLOCATION Performance Analysis of Concurrency Control Algorithms		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Jan 1981 - Jan 1983
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Wente K. Lin, Nathan Goodman, Philip A. Bernstein, Jerry Nolte		8. CONTRACT OR GRANT NUMBER(s) F30602-81-C-0028
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Corporation of America Four Cambridge Center Cambridge MA 02142		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55812121
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COTD) Griffiss AFB NY 13441		12. REPORT DATE October 1983
		13. NUMBER OF PAGES 128
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Emilie J. Siarkiewicz (COTD)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Databases
Concurrency Control
Reliability

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is the second of three volumes of the final technical report for the project "Distributed Database Control and Allocation." The first volume describes frameworks for understanding concurrency control and recovery algorithms. This volume describes work on the performance analysis of concurrency control algorithms. The third volume summarizes the results in the form of a distributed database designer's handbook.

This volume is a collection of five papers written during the course of the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) project, each paper analyzing from a different perspective the results of the performance study. The first paper presents a study that analyzes the relationship between the performance of the two phase locking algorithm and the following system parameters: access distribution of the database, data granularity, transaction size and multiprogramming level. In a distributed database system, communication delay is also a major factor affecting the performance of a concurrency control algorithm, and the second paper presents an analysis of the relationship between the performance of the two phase locking algorithm and the communication delay. Another important factor that affects the performance of a concurrency control algorithm is the number of read-only transactions relative to the number of write transactions -- ratio of read-only to write transactions. The third paper presents an analysis of the relationship between the performance of the two phase locking algorithm and their ratio.

The fourth paper extends the analysis to algorithms based on timestamps by presenting a comparison of the performance of three distributed concurrency control algorithms -- the Basic Timestamp, Multiple Version Timestamp, and two phase locking algorithm.

The fifth paper analyzes the two phase locking algorithm in more detail and refine the algorithm into nine algorithms. In addition, the previous two timestamp algorithms are reevaluated in more detail and analyze a new timestamp based algorithm -- the Dynamic Timestamp algorithm. Then the performance of the twelve algorithms is compared.

CONTENTS

SECTION I

INTRODUCTION

This is the second volume of the final technical report for the project "Distributed Database Control and Allocation," sponsored by Rome Air Development Center, contract number F30602-81-C0028. This volume describes work on the performance analysis of concurrency control algorithms.

This volume is a collection of papers written during the course of the project, each paper analyzing from a different perspective the results of the performance study. It consists of five sections. Section I presents a study that analyzes the relationship between the performance of the two phase locking algorithm and the following system parameters: access distribution of the database, data granularity, transaction size and multiprogramming level. In a distributed database system, communication delay is also a major factor affecting the performance of a concurrency control algorithm, and we present in Section II an analysis of the relationship between the performance of the two phase locking algorithm and the communication delay. Another important factor that affects the performance of a concurrency control algorithm is the number of read-only transactions relative to the number of write transactions — ratio of read-only to write transactions. In Section III we present an analysis of the relationship between the performance of the two phase locking algorithm and their ratio.

Section IV extends the analysis to algorithms based on timestamps by presenting a comparison of the performance of three distributed concurrency control algorithms — the Basic Timestamp, Multiple Version Timestamp, and two phase locking algorithm.

In Section V we analyze the two phase locking algorithm in more detail and refine the algorithm into nine algorithms. In addition, we reevaluate the previous two timestamp algorithms in more detail and analyze a new timestamp based algorithm — the Dynamic Timestamp algorithm. We then compare the performance of the twelve algorithms.

SECTION II

PERFORMANCE OF TWO PHASE LOCKING*

Wente K. Lin

Jerry Nolte

* This paper appeared in the Proceedings of the 1982 Berkeley Workshop on Distributed Data Management and Computer Networks.

2. Performance of Two Phase Locking

Abstract

Simulation and analytical modeling of the two phase locking in a DBMS is the subject of this study. It is only part of a larger project that is studying the performances of various concurrency control and reliability algorithms in a distributed DBMS. In the simulation model, the application environment is characterized by the transaction size -- the number of lockable units requested by each transaction -- and the system environment by the number of transactions running concurrently (multiprogramming level), total number of lockable units in the database, and the distribution of accesses to these lockable units. These environments are varied for different simrlation runs. Output from these simulation runs includes the probabilities of a lock request involved in a conflict and deadlock respectively (PC and PD), and the average waiting delay (WT) and its standard deviation (DV) of a blocked lock request. The results show that the system behaves quite similarly for different access distributions -- PC, PD, WT, and DV all increase more than linearly with the multiprogramming level and the transaction size; the increase of PC is faster with multiprogramming level than with the transaction size, and the reverse is true for PD, WT, and DV. Regression analysis on the simulation results reveals interesting relationships between the granularity of the lockable units and PC, PD, and WT. Because of the assumption of fixed delay (excluding blocking due to lock conflict) between two consecutive lock requests by a transaction, the results apply to a centralized DBMS with little IO delay variation, and a distributed DBMS with little communication delay variation.

2.1 Introduction

In the two phase locking protocol as described in Gray [1], during the first phase transactions accummulate locks incrementally, acquiring each lock as its need arises, and during the second phase, release each lock as soon as its need ends. But to spare the end users the responsibility of requesting and releasing locks, most DBMSs implement implicit locking. The DBMSs request and release the locks automatically when the transactions request the data items and when the transactions end, respectively. Because a DBMS, not knowing enough of the syntax and semantics of the transactions, is ignorant of the time when each data item is no longer needed, it can only release the locks held by a transaction when the transaction ends. Besides, if locks held by a transaction are released before the transaction ends, then the abortion of the transaction causes roll-backs of all other transactions that have read data released by the aborted transaction. To avoid the problems discussed above, most DBMS release locks held by a transaction when the transaction ends. The performance of this modified two-phase locking is the subject of this study.

In this study we use several measures of system performance. We emphasize the blocking and restart behavior of transactions. We concentrate on the basic underlying factors of conflict, deadlock, and wait duration. The performance variables are listed as follows:

1. the average probability of a lock request conflicting with another one;
2. the average probability of a lock request causing a deadlock;
3. the average waiting delay of a conflicting lock request;
4. and the standard deviation of this delay.

Besides locking protocol, the performance of a DBMS depends on several system and application parameters:

1. the average number of locks requested by a transaction (transaction size);
2. the maximum number of transactions running concurrently (the multiprogramming level);
3. the size of the group that is the unit of locking (lockable unit size);
4. the size of the database (total number of lockable units);
5. and the distribution of lock requests to the lockable units of the database.

Two distributions of lock requests to the lockable units are simulated. The random access model assumes that all lockable units have the same probability of being accessed by a lock request. The 20/80 model assumes that 20% of the database is accessed 80% of the time.

Using simulation and statistical data analysis techniques, this paper studies the relationships between the performance of a DBMS and those system and application parameters affecting it.

A few researchers have attempted similar studies. In Lin [2], the same approach taken in this study was used to evaluate two timestamping protocols, but its results could not be extended to the two-phase locking protocol. In Naka [3], the result confirmed that concurrent updating of the database by transactions degrades the performance of a DBMS. In Spit [4], the two phase locking and the modified version (described above) were found to perform equally well in system-2000. In Mun [5], deadlock resolution methods were studied, and three were found to be superior: restarting the smallest, the one holding the least locks, and the one having consumed the least cpu time. In addition, it was found that simultaneous reduction of the sizes of the lockable unit and the transaction improves the performance. But the oversimplified definition of performance as the cpu utilization made the results less useful. In Ries [6], the scope and the objective of its simulation were much more ambitious than the previous three. Nevertheless, it emphasized the effects of the size of the lockable unit on the performance of the DBMS, which was defined as the cpu and IO utilizations, plus in some cases the response time and the system through-put. The main model required transactions to obtain all the required locks before they started, and the request-as-needed model was only briefly studied. It had many interesting results showing how the size of the lockable unit interacts with the system and application parameters to effect the performance. But its assumption that the multiprogramming level has no affect on performance is contradicted by this study. Also, performance was not related to system and application parameters as precisely and quantitatively as in the present study.

This study expands on Lin [2] and Ries [4], and presents the results in the same precise form as that of Lin [2]. The second subsection discusses the simulation model; the third subsection presents and analyzes the results of the random access model; the fourth subsection presents and summarizes the results of the 20/80 model; and the fifth subsection summarizes the results of this study.

2.2 Simulation Model

A complete description of a simulatio model for a DBMS must include the database, the transactions, mputer system, and the output parameters.

The database consists of DZ (Database siZe) lockable units of equal size. The size of each lockable unit is irrelevant to our model. The database size DZ varies among different simulation runs.

We simulate two different access distributions to the database: the random access model in which all lockable units are equally likely to be accessed, and the 20/80 access model in which 20% of the database is accessed 80% of the time.

All transactions request only exclusive locks. Within each simulation run, all transactions request the same number TZ (Transaction siZe) of lockable units, but TZ varies among different simulation runs. Each transaction requests its lockable units sequentially, but different transactions request lockable units asynchronously. When a transaction requests for a lockable unit, a random number is drawn to select one among all the lockable units in the database except those held by the requesting transaction; thus a transaction never requests the same lockable unit more than once. If the drawn lockable unit is locked by another transaction, the requesting transaction is queued at the end of a FIFO queue. Otherwise, it sets a lock on the drawn lockable unit and waits one time unit before requesting another lockable unit. Since processing a lock request is assumed to be instantaneous, the simulation timer is advanced one unit only after all outstanding lock requests have been processed. The assumption that a transaction waits a unit of time (after obtaining a lockable unit) before requesting another one, implies that it takes one time unit to retrieve a lockable unit from the database, to wait for the cpu, and to process it. Each transaction releases all its lockable units after its completion or abortion.

We model the computer system at a high functional level. The cpu, IO devices, and other hardware components are invisible in the simulation model; their existence is implied by the processing time required for each lockable unit discussed previously. The system is a closed multiprogramming system, i.e., the number of transactions running concurrently remains at a constant level MP (MultiProgramming level); a new transaction starts as soon as one completes or aborts. Nonetheless MP varies among different simulation runs. A lock request conflicts if it requests a lockable unit already held by another transaction. The system maintains a lock with a FIFO queue for each lockable unit and places conflicting lock requests into the queue. It checks for deadlocks as soon as a lock request conflicts. If it detects a deadlock, the transaction of the conflicting lock request aborts and restarts immediately; it restarts with a new randomly drawn sequence of lock requests. Checkings of conflicts and deadlocks are instantaneous.

For each simulation run, the output includes the fraction of conflicting lock requests (which is the same as the probability of a lock request conflicting with another lock request PC), the fraction of conflicting lock requests causing deadlocks (which is the same as the probability of a lock request causing a deadlock PD), and the average waiting of a blocked lock request (WT) and its standard deviation (DV).

2.3 Simulation Results of the Random Access Model

Sixty four simulations were run for 4 values of multiprogramming level (MP), transaction size (TZ), and database size (DZ) each. The results are presented and analyzed in this subsection in the following order: PC, PD, WT, and DV. The analysis consists of three steps: visual inspection, regression analysis, and examination of the regression equations.

The results of PC are presented in Figure 2.1. The figure shows that for a fixed DZ, PC increases with both MP and TZ, and the increase is larger with MP than with TZ. This behavior is explained by the following observation during the simulation runs: the number of transactions deadlocked increases faster with the transaction size than with the multiprogramming level. Since a deadlocked transaction aborts and releases all held locks as soon as the deadlock occurs, the total number of locks outstanding (not released) increases slower with the transaction size than with the multiprogramming level.

If a diagonal line is drawn from the top left to the bottom right of each table in the figure, each number below the line is always larger than the opposite number across the line. Assuming DZ is fixed, two elements across the diagonal line represent the same load (L) defined as the product of MP and TZ divided by DZ. For example, a system with 16 transactions, each requesting 7 locks, imposes the same load (112 lockable units) on the database as a system with 7 transactions, each requesting 16 locks. This line shows that with the same load, the system with higher multiprogramming level has higher probability of conflict than the system with higher transaction size. This behavior is explained by the following observation during the simulation runs. Assuming the load L and the database size DZ are fixed, then on the average, a larger MP with smaller TZ implies less deadlocks and more locks outstanding. Since each lockable unit has the same probability of being accessed, more outstanding locks means higher probability of conflict. But higher probability of conflict does not necessarily means longer response time, because smaller transaction size causes conflicting requests to wait less and to deadlock less, as will be shown.

The differences across the diagonal line diminish as the database size DZ increases -- that is, the probability of conflict (PC) is approximately proportional to the load L when the load on the database is light, because increasing the database size without increasing the multiprogramming level or the transaction size is equivalent to decreasing the load on the database.

We applied regression analysis to the data in Figure 2.1, and found equation (2.1) a good fit. The residuals -- the differences between the actual values and the values predicted by the equation -- are within 2.5% of the actual values. We did a few simulation runs with larger values of DZ, MP, and TZ, and found that the equation is still a good fit for DZ of up to 12384, MP of up to 128, and TZ of up to 32; but we found that when the transaction size TZ gets much larger than 32, the equation under-estimates the probability of conflict (PC) substantially.

$$PC = \frac{0.72\,(MP-1)^{1.05+0.35L}\,TZ^{1.08-0.13L}}{DZ^{1.08+0.28L}} \qquad (2.1)$$

$$L = \frac{MP\ TZ}{DZ}$$

Next, we use the regression equation to examine the relationship between the size of the lockable unit and the probability of conflict.

If we split each lockable unit into k smaller units, then the database size increases to k times its original size. Because of the smaller lockable units, a transaction must request more lockable units; thus the transaction size increases to w ($1 \leq w \leq k$) times its original size. The value of w depends on how well the database is placed before the split. If the database is originally well placed, then all the data items contained in the original TZ lockable units are wanted by the transaction -- no frivolous data items are retrieved. In this case, when a lockable unit is split into k smaller ones, the transaction size increases to k times its original size (w=k). Otherwise, if the database is badly placed before the split, then the lockable units retrieved by a transaction contain a lot of unwanted data items. Thus, after the split, a transaction may request the same number of lockable units and still obtain all the data items it needs (w=1). In most cases, however, w will be larger than one and smaller than k.

Replacing DZ and TZ by kDZ and wTZ, equation (2.1) becomes equation (2.2),

$$PC' = t \times PC \qquad (2.2)$$

where

$$t = \frac{DZ^{0.28Lr}\ TZ^{0.13Lr}\ w^{1-(0.13Lw)/k}}{(MP-1)^{0.35Lr}\ k^{1-(0.28Lw)/k}} \qquad (2.2a)$$

and

$r = (k-w)/k$.

Setting w to k, equation (2.2a) becomes (2.2b).

$$t = \frac{1}{k^{0.41L}} \tag{2.2b}$$

Since k is larger than one, t is smaller than one. Thus smaller lockable units imply a smaller probability of conflict whenever the database is well placed. But as we will show later, smaller probability of conflict with larger transaction size may result in a higher probability of deadlock and longer transaction response time. As L approaches zero, i.e., the load is light, t approximates one, and the difference between PC and PC′ becomes insignificant.

Setting w to one in equation (2.2a) results in equation (2.2c).

$$t = \frac{DZ^{0.28Lr}\ TZ^{0.13Lr}}{(MP-1)^{0.35Lr}\ k^{1-(0.28L)/k}} \tag{2.2c}$$

where

$r = (k-1)/k$

and

$t = 1/k$ as L approaches zero.

Equation (2.2c) shows that when the load L is smaller than 100%, which is within our simulation range and is realistic, t is less than one. Therefore, if the database is badly placed, smaller lockable units imply a smaller probability of conflict. In this case, since the transaction size remains the same, a smaller probability of conflict does imply a smaller probability of deadlock and shorter response time.

To sum up, smaller lockable units always imply smaller probability of conflict.

The probabilities of deadlock (PD) are presented in Figure 2.2. Notice that PD is the conditional probability of a lock request causing a deadlock, given that the request conflicts. The unconditional probability of deadlock is the product of PC and PD, which is presented in Figure 2.3. These data are also analyzed in three steps: visual inspection, regression analysis, and analysis of the regression equation.

Figure 2.3 shows that for a fixed DZ, PD increases with both the multiprogramming level MP and the transaction size TZ. But in contrast to PC, the increase is larger with TZ than with MP.

If the diagonal line discussed previously is drawn for each table in Figure 2.3, the number below the line is always smaller than the corresponding number across the line, in sharp contrast to PC of Figure 2.1. Thus assuming equal loads L, a system with larger transactions and lower multiprogramming level has a higher probability of deadlock than a system with shorter transactions and higher multiprogramming level.

Similarly, regression analysis shows equation (2.3) a good fit for the data of Figure 2.3.

$$PD' = PDxPC = \frac{0.012(MP-1)^{1.07-0.24L}\ TZ^{3.61-3.48L}}{DZ^{1.99-1.79L}} \tag{2.3}$$

$$L = \frac{MP \times TZ}{DZ}$$

We must emphasize that PD is the probability of deadlock for a lock request, not a transaction. Equation (2.3) shows that when the load L is larger than 80%, the coefficient c is smaller than the coefficient b. Therefore, for a fixed load of 80% or greater, a system with shorter transactions and higher multiprogramming level has a higher probability of deadlock than a system with longer transactions and lower multiprogramming level. This rather surprising behavior is not immediately apparent from inspection of Figure 2.3. This behavior occurs because when the load is high and transactions are long, transactions deadlock and abort frequently; and abortions of long transactions means that more locks are freed. Thus there is less probability of a lock request causing a deadlock.

To analyze the relationship between PD and the lockable unit size, we replace DZ by kDZ and TZ by wTZ, and equation (2.3) becomes equation (2.4).

$$PD'' = t \times PD' \tag{2.4}$$

where

$$t = \frac{(MP-1)^{0.54Lr}\ TZ^{3.7Lr}\ w^{3.5-(3.7Lw)/k}}{DZ^{2.1Lr}\ k^{1.9-(2.1Lw)/k}} \tag{2.4a}$$

and

$$r = (1-w/k)$$

Setting w to k, equation (2.4a) becomes (2.4b), which shows that if and only if the load L is less than one, which is within the range of our simulation and is realistic, t is greater than one.

$$t = k^{1.6(1-L)} \qquad (2.4b)$$

Thus, when the database is well placed, smaller lockable units imply a larger probability of deadlock.

Setting w to 1 for the originally badly placed system, equation (2.4a) becomes (2.4c), which shows that, within the range of our simulation, t is less than one. Therefore smaller lockable units reduce the probability of deadlock.

In summary, larger lockable units in a well placed system and smaller lockable units in a badly placed system reduce the probability of deadlock for lock requests and transactions.

$$t = \frac{(MP-1)^{0.54Lr}\ TZ^{3.7Lr}}{DZ^{2.1Lr}\ k^{1.9-(2.1L)/k}} \qquad (2.4c)$$

where

$r = (1-1/k).$

The average waiting times of a conflicting lock request are shown in Figure 2.4, which shows that the average waiting of a conflicting lock request increases with the multiprogramming level and the transaction size, and the increase is larger with the transaction size than with the multiprogramming level. The result is consistent with our intuition, because a lock request blocked by a long transaction must wait until the long transaction completes or aborts; and it takes longer for a long transaction to complete or abort. Also, if a similar diagonal line is drawn for each table, the number above the line is always larger than the corresponding number across the diagonal line.

Regression analysis shows equation (2.5) a good fit for the data of Figure 2.4.

$$WT = \frac{0.19(MP-1)^{3.4(L+0.2)^2-0.3}\ TZ^{2.7(L+0.15)^2+0.8}}{DZ^{4.1(L-0.04)^2-0.16}} \qquad (2.5)$$

Assuming the database is well placed, to reduce the granularity of the lockable units to 1/k of its original size, we increase the database size DZ and transaction size TZ to kDZ and kTZ respectively in equation (2.5), resulting in equation (2.6a). Equation (2.6a) shows that when the load L is less than 1.4, which is realistic and within the range of our simulations, smaller lockable units imply longer waiting for a conflicting lock request. The result is consistent with the earlier observation — longer transactions induce longer waiting.

$$WT = k^{1.25-1.37(L-0.41)^2} \qquad (2.6a)$$

Assuming the database is badly placed, to reduce the granularity of the lockable units to 1/k of its original size we increase the database size DZ to kDZ, but leave the transaction size TZ unchanged in equation (2.5), resulting in equation (2.6b). Equation (2.6b) shows that when the load is light and k is small, t is greater than one — longer waiting for a conflicting lock request. As shown earlier this is because when a database is badly placed and the load is light, reducing the size of the lockable units reduces the probability of deadlock. With less deadlocks, more transactions complete and less transactions abort. Since a transaction takes longer to complete than to abort, a blocked lock request waits longer.

$$WT = \frac{DZ^{4.1rL(qL-0.08)}}{(MP-1)^{3.4rL(qL+0.4)}\,TZ^{2.7rL(qL+0.3)}\,k^{4.1(L/k-0.04)^2-0.16}} \qquad (2.6b)$$

where

$$r = (1 - 1/k)$$
$$q = (1 + 1/k)$$

In summary, whether the database is well placed or badly placed, smaller lockable units increase waiting delay for a blocked lock request, except when load is extremely heavy, the database is badly placed, and the reduction in lockable unit size is large.

We next examined the standard deviation of waiting delays. These results can be summarized very simply.

Regression on the data of Figure 2.5 results in equation 2.7, which shows that the waiting delay may be approximated by an Erlangian distribution.

DV = 0.86 x WT (2.7)

DZ = 256

MP/TZ	7	10	12	16
7	.077	.104	.118	.135
10	.113	.145	.159	.176
12	.135	.169	.182	.198
16	.174	.210	.224	.236

DZ = 1025

MP/TZ	7	10	12	16
7	.020	.029	.034	.045
10	.030	.043	.050	.064
12	.037	.052	.061	.076
16	.050	.069	.081	.098

DZ = 512

MP/TZ	7	10	12	16
7	.040	.056	.066	.081
10	.059	.081	.094	.112
12	.072	.097	.111	.130
16	.096	.127	.142	.160

DZ = 2048

MP/TZ	7	10	12	16
7	.010	.015	.017	.023
10	.015	.022	.026	.034
12	.019	.026	.031	.041
16	.025	.036	.043	.055

PC : Probability of a Lock Request conflicting With Another Lock Request

Figure 2.1

DZ = 256

MP/TZ	7	10	12	16
7	.031	.078	.112	.183
10	.039	.102	.143	.207
12	.044	.115	.156	.218
16	.061	.141	.179	.232

DZ = 1024

MP/TZ	7	10	12	16
7	.006	.014	.026	.050
10	.008	.019	.028	.061
12	.007	.019	.033	.068
16	.007	.025	.040	.090

DZ = 512

MP/TZ	7	10	12	16
7	.014	.037	.052	.102
10	.014	.041	.068	.130
12	.017	.049	.079	.144
16	.021	.067	.102	.168

DZ = 2048

MP/TZ	7	10	12	16
7	.003	.006	.011	.024
10	.003	.006	.011	.024
12	.003	.009	.014	.029
16	.003	.009	.015	.034

PD : Conditional Probability of a Lock Request Causing a Deadlock after Conflict

Figure 2.2

DZ = 256

MP/TZ	7	10	12	16
7	.0024	.0081	.0132	.0247
10	.0044	.0148	.0227	.0364
12	.0059	.0194	.0284	.0432
16	.0106	.0296	.0401	.0548

DZ = 1024

MP/TZ	7	10	12	16
7	.00012	.00040	.00088	.00225
10	.00024	.00081	.00140	.00390
12	.00026	.00099	.00201	.00517
16	.00035	.00173	.00324	.00882

DZ = 512

MP/TZ	7	10	12	16
7	.0006	.0021	.0034	.0083
10	.0008	.0033	.0064	.0146
12	.0012	.0048	.0088	.0187
16	.0020	.0085	.0145	.0269

DZ = 2048

MP/TZ	7	10	12	16
7	.000030	.00009	.00019	.0006
10	.000045	.00013	.00029	.0008
12	.000057	.00023	.00043	.0012
16	.000075	.00032	.00063	.0019

PCxPD : Absolute Probability of a Lock Request Causing a Deadlock after Conflict

Figure 2.3

DZ = 256

MP/TZ	7	10	12	16
7	3.76	6.18	7.85	11.01
10	4.64	8.25	10.55	14.40
12	5.36	9.52	12.09	15.70
16	7.27	12.52	15.24	18.65

DZ = 1024

MP/TZ	7	10	12	16
7	3.09	4.49	5.60	8.26
10	3.19	4.93	6.42	10.57
12	3.35	5.34	7.25	11.80
16	3.54	6.65	9.30	16.05

DZ = 512

MP/TZ	7	10	12	16
7	3.33	5.12	6.60	9.72
10	3.66	6.18	8.50	13.37
12	3.88	7.19	9.91	15.28
16	4.71	9.77	13.53	19.43

DZ = 2048

MP/TZ	7	10	12	16
7	2.94	4.11	5.00	7.01
10	3.01	4.39	5.42	8.13
12	3.07	4.49	5.64	8.88
16	3.14	4.88	6.35	10.91

WT : Average Waiting Time of a Conflicting Lock Request after the Conflict

Figure 2.4

2.4 Results of 20/80 Access Model

The results of simulating the 20/80 access model are shown in Figures 2.6 through 2.10. They are similar to the results of the random access model with heavier load. The reason is that when 20% of the database is used 80% of the time, the same load of the random access model becomes a heavier load. The probability of conflict, the probability of deadlock, and the average waiting of a conflicting lock request still increases with both the transaction size and the multiprogramming level.

DZ = 256

MP/TZ	7	10	12	16
7	2.86	5.28	6.90	10.09
10	4.02	7.59	9.88	13.56
12	4.93	9.03	11.26	14.78
16	7.05	11.77	14.19	17.66

DZ = 1024

MP/TZ	7	10	12	16
7	1.95	3.35	4.36	6.92
10	2.16	3.94	5.50	9.62
12	2.38	4.52	6.49	10.98
16	2.68	6.15	8.97	15.51

DZ = 512

MP/TZ	7	10	12	16
7	2.29	4.08	5.44	8.61
10	2.78	5.45	7.76	12.51
12	7.19	6.71	9.22	14.32
16	4.32	9.45	12.90	18.07

DZ = 2048

MP/TZ	7	10	12	16
7	1.80	2.80	3.49	5.46
10	1.89	3.09	4.06	6.93
12	1.94	3.21	4.53	8.04
16	2.09	3.92	5.52	10.58

DV : Standard Deviation of the Waiting Times of Conflicting Lock Requests

Figure 2.5

The probability of conflict increases faster with the multiprogramming level than with the transaction size, while the reverse is true for the probability of deadlock and the average waiting of a conflicting lock request. If diagonal lines are drawn for the tables (as previously explained), the number below the line is always larger than the corresponding number above the line for the probability of conflict, and the opposite is true for the probability of deadlock and the average waiting of a conflicting lock request. But the differences diminish as the load becomes lighter.

Applying regression analysis to data in Figure 2.6 results in equation (2.8). Similar to equation (2.1), it shows that the coefficient b is always larger than the coefficient c. The major difference between this equation and equation (2.1) is that the coefficient a of equation (2.8) is equal 2.7, much larger than the 0.72 of equation (2.1).

$$PC = \frac{3.7(MP-1)^{1.08+1.51L}\ TZ^{1.08+0.58L}}{DZ^{1.13+1.39L}} \qquad (2.8)$$

where

$$L = \frac{MP \times TZ}{DZ}$$

To examine the relationship between the probability of conflict and the lockable unit size, we replace TZ by wTZ and DZ by kDZ in equation (2.8), and obtain equation (2.9).

$$PC' = t \times PC \tag{2.9}$$

where

$$t = \frac{DZ^{1.39rL}\, w^{1+(0.58Lw)/k}}{TZ^{0.58rL}\, MP^{1.51rL}\, k^{1+(1.39Lw)/k}} \tag{2.9a}$$

and

$$r = 1 - w/k.$$

If the database is well placed, then w is equal to k, and equation (2.9a) becomes equation (2.9b), which shows that smaller lockable units reduce the probability of conflict, consistent with the result of the random access case.

$$t = k^{-0.81} \tag{2.9b}$$

If the database is badly placed, then w is equal to one, and equation (2.9a) becomes equation (2.9c). Equation (2.9c) shows that if the load L is less 50%, which is within the range of our simulations and is realistic, smaller lockable units reduce probability of conflict. In summary, whether the database is originally well or badly placed, reducing lockable units reduces the probability of conflict. This result is the same as in the random access model.

$$t = \frac{DZ^{1.39rL}}{TZ^{0.58rL}\, MP^{1.51rL}\, k^{1+(1.39L)/k}} \tag{2.9c}$$

where

$$r = 1 - 1/k.$$

Regression of the data in Figure 2.8 results in equation (2.10), which shows that when the load L is greater than 33%, the coefficient c is smaller than the coefficient b. Therefore, for a fixed load of 33% or higher, a system with higher multiprogramming level and smaller transactions has higher probability of deadlock than a system with lower multiprogramming level and longer transactions. This result is similar to the random access model.

$$PD' = PD \times PC = \frac{0.8(MP-1)^{1.41+2.66L}\, TZ^{3.88-4.74L}}{DZ^{2.33-0.13L}} \qquad (2.10)$$

where

$$L = \frac{MP \times TZ}{DZ}$$

To examine the relationship between the probability of deadlock and the lockable unit size, we replace TZ by wTZ and DZ by kDZ in equation (2.10), and obtain equation (2.11).

$$PD'' = t \times PD' \qquad (2.11)$$

where

$$t = \frac{TZ^{4.74Lr}\, w^{3.88-(4.74Lw)/k}}{(MP-1)^{2.66Lr}\, DZ^{0.13Lr}\, k^{2.33-(0.13Lw)/k}} \qquad (2.11a)$$

If the database is well placed, then w is equal to k, and equation (2.11a) becomes equation (2.11b). Similar to equation (2.4b), it shows that when the load L is less than 34%, which is realistic and within the range of our simulations, t is greater than one. That means larger lockable units reduce the probability of deadlock. This result is similar to the one found in the random access model.

$$t = k^{1.55-4.61L} \qquad (2.11b)$$

For the badly placed database, setting w to one in equation (2.11a) results in equation (2.11c), which shows that, within the range of our simulations, smaller lockable units reduce the probability of deadlock. This result is also similar to the one found in the random access model.

$$t = \frac{TZ^{4.74Lr}}{(MP-1)^{2.66Lr}\, DZ^{0.13Lr}\, k^{2.33-(0.13L)/k}} \qquad (2.11c)$$

Regression on the data in Figure 2.9 results in equation (2.12).

$$WT = \frac{0.037(MP-1)^{11.7(L-0.1)^2-0.24}\, TZ^{14.8(L-0.22)^2+0.25}}{DZ^{13.4(L-0.2)^2-0.27}} \qquad (2.12)$$

Replacing DZ by kDZ and TZ by kTZ, equation (2.12) becomes equation (2.13a), which shows, as does equation (2.6a), that t is greater than one — longer waiting delay for a conflicting lock request.

$$WT = k^{0.1+1.4(L-0.4)^2} \tag{2.13a}$$

Replacing DZ by kDZ, but leaving TZ unchanged, equation (2.12) becomes equation (2.13b), which shows, as does equation (2.6b), that when the load is light and k is small, t is greater than one. Therefore, in general, reducing the size of lockable units increases the waiting delay of a conflicting lock request, except when the load is heavy, the database is badly placed, and the reduction of lockable unit size is large.

$$WT = \frac{DZ^{13.4rL(qL-0.4)}}{(MP-1)^{11.7rL(qL-0.2)}\,TZ^{14.8rL(qL-0.44)}\,k^{13.4(L/k-0.2)^2-0.27}} \tag{2.13c}$$

where

$r = (1 - 1/k)$
$q = (1 + 1/k)$

Regression on the data in Figure 2.10 results in equation (2.14).

$$DV = -0.88 + WT \tag{2.14}$$

DZ = 512

MP/TZ	7	10	12	16
7	.119	.150	.163	.176
10	.166	.198	.210	.221
12	.192	.225	.237	.245
16	.236	.268	.277	.284

DZ = 2048

MP/TZ	7	10	12	16
7	.033	.045	.054	.067
10	.048	.067	.078	.095
12	.059	.080	.092	.110
16	.078	.105	.119	.137

DZ = 1024

MP/TZ	7	10	12	16
7	.063	.085	.098	.116
10	.093	.122	.135	.151
12	.110	.142	.155	.173
16	.143	.177	.191	.207

DZ = 4096

MP/TZ	7	10	12	16
7	.016	.024	.028	.036
10	.025	.035	.041	.053
12	.030	.042	.050	.064
16	.040	.057	.067	.083

PC : Probability of a Lock Request conflicting With Another Lock Request

Figure 2.6

DZ = 512

MP/TZ	7	10	12	16
7	.057	.124	.168	.230
10	.072	.149	.189	.242
12	.081	.161	.201	.246
16	.102	.181	.214	.247

DZ = 2048

MP/TZ	7	10	12	16
7	.011	.028	.044	.086
10	.012	.032	.050	.104
12	.013	.035	.060	.113
16	.015	.046	.078	.141

DZ = 1024

MP/TZ	7	10	12	16
7	.025	.057	.089	.156
10	.028	.075	.117	.181
12	.032	.086	.126	.195
16	.042	.109	.149	.211

DZ = 4096

MP/TZ	7	10	12	16
7	.005	.012	.019	.037
10	.006	.013	.021	.046
12	.005	.014	.023	.051
16	.005	.016	.029	.067

PD : Conditional Probability of a Lock Request
Causing a Deadlock after Conflict

Figure 2.7

DZ = 512

MP/TZ	7	10	12	16
7	.0068	.0186	.0274	.0405
10	.0120	.0295	.0397	.0535
12	.0156	.0362	.0476	.0603
16	.0241	.0485	.0593	.0701

DZ = 2048

MP/TZ	7	10	12	16
7	.00036	.00126	.00238	.00576
10	.00058	.00214	.00390	.00988
12	.00077	.00280	.00552	.01243
16	.00117	.00483	.00928	.01931

DZ = 1024

MP/TZ	7	10	12	16
7	.0016	.0048	.0087	.0181
10	.0026	.0092	.0158	.0273
12	.0035	.0122	.0195	.0337
16	.0060	.0193	.0285	.0437

DZ = 4096

MP/TZ	7	10	12	16
7	.00008	.00028	.00053	.00133
10	.00015	.00045	.00086	.00243
12	.00015	.00058	.00115	.00326
16	.00020	.00091	.00194	.00556

PCxPD : Absolute Probability of a Lock Request
Causing a Deadlock after Conflict

Figure 2.8

DZ = 512

MP/TZ	7	10	12	16
7	4.21	6.81	9.49	11.11
10	5.44	8.94	10.83	13.08
12	6.50	10.19	11.93	14.39
16	8.44	12.64	14.26	16.32

DZ = 2048

MP/TZ	7	10	12	16
7	3.22	4.81	6.11	9.36
10	3.50	5.70	7.69	12.36
12	3.65	6.32	8.99	14.5
16	4.19	8.50	11.92	18.43

DZ = 1024

MP/TZ	7	10	12	16
7	3.51	5.71	7.37	10.83
10	4.25	7.54	9.75	13.89
12	4.75	8.71	11.43	15.79
16	6.04	11.49	14.53	18.92

DZ = 4096

MP/TZ	7	10	12	16
7	3.07	4.33	5.28	7.66
10	3.15	4.65	5.96	9.49
12	3.21	5.00	6.58	10.87
16	3.40	5.77	7.99	14.54

WT : Average Waiting Time of a Conflicting Lock Request after the Conflict

Figure 2.9

DZ = 512

MP/TZ	7	10	12	16
7	3.48	5.94	8.49	10.35
10	4.89	8.36	10.14	12.49
12	6.04	9.62	11.38	13.75
16	7.98	11.93	13.54	15.65

DZ = 2048

MP/TZ	7	10	12	16
7	2.17	3.75	5.00	8.01
10	2.57	4.83	6.95	11.63
12	2.79	5.69	8.45	13.70
16	3.56	8.22	11.47	17.18

DZ = 1024

MP/TZ	7	10	12	16
7	2.60	4.78	6.41	9.72
10	3.59	6.96	9.24	13.02
12	4.21	8.18	10.72	14.92
16	5.73	10.98	13.72	17.68

DZ = 4096

MP/TZ	7	10	12	16
7	1.91	3.05	3.97	6.26
10	2.07	3.53	4.83	8.55
12	2.17	4.02	5.72	10.25
16	2.48	5.08	7.61	14.02

STD-DEV : Standard Deviation of the Waiting Times of Conflicting Lock Requests

Figure 2.10

2.5 Summary

We simulated the two-phase locking in a DBMS with fairly constant communication and IO delays. We collected performance data, and regressed these data into equations relating the performance of the DBMS to the multiprogramming level, the transaction size, and the database size. Using these equations we examine the interaction between the performance of a DBMS and lockable units size.

We found the performance behavior of a DBMS with random database access distribution quite similar to that of the 20/80 access distribution — the 20/80 system behaves as a random access system in heavy load. In fact, the same regression models (equations) with different coefficient values fit both access models well except for the standard deviation of the lock request waiting delay.

The probability of conflict of a lock request increases more than linearly with the multiprogramming level and the transaction size; the increase is larger with the multiprogramming level than with the transaction. The probability of deadlock, the average waiting, and its standard deviation of a conflicting lock request also increase more than linearly with the multiprogramming level and the transaction size. But the increase is smaller with the multiprogramming level than with the transaction size.

The waiting delay of a conflicting lock request can be approximated by an Erlangian distribution in the random access model. This result can be extremely useful for researchers who use queueing theory to model a DBMS.

The results of this study have been validated, and can be extrapolated for database size of up to 12384, multiprogramming level of up to 128, and transaction size of up to 32.

So far we have concentrated on the basic factors of PC, PD, WT, and DV. We will next briefly discuss the combination of these blocking and restart variables into system throughput, a measure of performance which is more directly useful to a system designer.

In the highly functional model used here, all system resources are represented by the time to process lock requests. Since each request consumes the same time, we measure throughput by number of lock requests processed by transactions which finish.

In every case, throughput decreases with increasing TZ, if MP and DZ are held constant. As noted above, for longer transactions there are more conflicts, more deadlocks, and longer delays. The message for applications program design is clear. Transactions should be made as small as possible.

Also, throughput increases with increasing DZ if MP and TZ are held constant. This is the "badly placed locks" case, and it also can be anticipated from the analysis above. For random access of data, small granules will provide better throughput when both blocking and restart behavior are considered. However, because of the increasing communications and processing costs of lock management, the response time will increase. The optimal granularity can be calculated from the regression equations.

Finally, throughput first increases, and then decreases with increased MP if TZ and DZ are constant. Given a particular granularity and transaction size, for light loads, significant gains in throughput can be attained by increasing the multiprogramming level. However, as the system load becomes heavier, the losses to deadlock and restart more than outweigh the gains from increased concurrency.

2.6 References

[1] Gray, J.N., et al., "Granularity of Locks and Degrees of Consistency in a Shared Data Base", Proc. IFIP Working Conference on Modelling of Data Base Management Systems, Freudenstadt, Germany, January 1976.

[2] Lin, W.T.K., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed DBMS", ACM-SIGMOD 1981 International Conference on Management of Data, Ann Arbor, Michigan, April 1981.

[3] Nakamura, et al., "A Simulation Model for a Database System Performance Evaluation", AFIPS Proc. 1975 NCC Conference, Volume 44, May 1975.

[4] Spitzer, J.F., "Performance Prototyping of Data Management Applications", Proc. CAM'76 Annual Conference, Houston, Texas, October 1976.

[5] Munz, R., et al., "Concurrency in Database System - A Simulation Study", Proc. ACM SIGMOD International Conference, Toronto, August 1977.

[6] Ries, D., "The Effect of Concurrency Control on Database Management System Performance", Ph.D. thesis, Electronics Research Lab, University of California, Berkeley, 1979.

SECTION III

COMMUNICATION DELAY AND TWO PHASE LOCKING*

Wente K. Lin

Jerry Nolte

* This paper appeared in the Third Int'l Conference on Distributed Computing Systems (Oct. 1982).

3. Communication Delay and Two Phase Locking

Abstract

A lock request in a distributed DBMS incurs many kinds of delay. We group these delays into two classes. The first one, called blocking delay, results from one lock request waiting for another lock request because of lock conflict. It is a direct result of concurrency control. The second delay, called communication delay, consists of communication network delay, IO delay, and CPU processing delay. In this paper, for two phase locking in a distributed DBMS, we study how the communication delay affects the blocking delay and system performance. The results show that the communication delay has little effect on the probability of conflict and deadlock of lock requests. The results also show that the blocking delay has a 2-stage erlagian distribution and that the number of locks held by a transaction when the transaction deadlocks and aborts has a 2-order negative binomial distribution. The results are important to performance modeling of distributed two phase locking algorithms, because they simplify the models.

3.1 Introduction

Many distributed concurrency control algorithms have been proposed (Bad[1], Ber[1], Ell[1], Gar[1], Lin[4], Ros[1], Ste[1], Sto[1], Tho[1]). But how well do they perform? A few researchers have attempted to compare the performance of different algorithms (Gar[1], Lin[1], Lin[2], Rie[1], Tha[1]). Unfortunately they all used different assumptions about system and application parameters. Furthermore, they compared different algorithms using different performance measures. For example, in Lin[1], two timestamping algorithms are compared, and the performance measure used is the average response time.

In Rie[1], two two-phase locking algorithms — the centralized method and the primary copy method — are compared. Performance measures include utilization of devices, average transaction response time, and others. In addition, a transaction must obtain all its locks before it can start, no duplicate data is allowed, and multiprogramming level is assumed to have no effect on system performance.

In Gar[1], three two-phase locking algorithms are compared — the centralized method, the voting method, and the ring method. Both utilization of devices and average response time are performance measures. A few major assumptions are made: multiprogramming level is one at each node, a transaction must obtain all locks before it proceeds, and a transaction requests all update locks in parallel. In addition, the main results apply to a fully replicated system.

In Lin[2], distributed two-phase locking algorithms are abstracted and encapsulated in one model, and the relation between system blocking behavior (conflict and deadlock) and various system and application parameters is studied in detail. Two access distributions of the database are simulated. The first one has a uniform distribution — every data granule is equally likely to be accessed by a lock request; the second one has 80% of the database accessed by 20% of the lock requests. It concludes that the more concentrated database access distribution has the same effect on system blocking behavior as the uniform distribution in heavier load.

In Tha[1], two two-phase locking algorithms are simulated — basic two-phase and centralized two-phase. The performance measure used is the average transaction response time. The simulation model includes many system and application parameters, thus necessitating a large number of simulation runs. But only the results of a few simulation runs with limited values of these parameters are presented.

These performance studies are very difficult to compare, and it is almost impossible to integrate their results. They compare different algorithms, they make different assumptions about system and application environments, and they employ different measures for system performance. This paper is part of a major effort to compare the principal distinct concurrency control algorithms, using the same performance measures and the same assumptions that are consistent with various system and application environments.

This paper reports part of the results of simulation on two-phase locking. Section 3.2 describes the simulation model, Section 3.3 discusses the simulation results, and Section 3.4 concludes the study.

3.2 The Simulation Model

Two phase locking causes blocking and deadlocks among transactions. Blocking occurs because two or more transactions may request the same data item at the same time. Blocking degradates system performance because a blocked lock request must wait for the blocking transaction to complete or abort. This waiting is called blocking delay. Deadlock occurs when two or more transactions directly or indirectly block each other. Deadlock also degradates system performance, becuase a deadlock causes a partially completed transaction to abort and restart.

Transaction blocking and restarting are affected by many system and application characteristics. These include average transaction size (number of locks requested by a transaction), multiprogramming level (number of transactions running concurrently), database size (number of locking granules), access distribution of the database (probability of each data granule being accessed by a lock request), frequency of local and remote requests, locking granularity, communication network, IO devices, memory size, CPU speed, and others. Thus to accurately evaluate the restarting and blocking behavior of the two phase locking, we must include all these factors in the simulation model, and this is too expensive to do directly.

To simplify the simulation, we model the system and transactions in a highly functional model. Much of the detail of a real distributed system is captured in a few parameters, which are used as inputs to the simulation model. This approach permits us to greatly reduce the number of simulation runs necessary, and also to reduce the complexity of the model, while retaining most of the impact that these details have on the performance of the concurrency control algorithms.

We model a transaction as a sequence of lock requests. Between two consecutive lock requests, a transaction incurs two kinds of delays: blocking delay and communication delay. The communication delay includes communication network delay, IO delay, and CPU processing delay; it is called communication delay because in a distributed system it is likely that the communication network delay dominates the IO and the CPU delays. The communication delay is an input parameter to our simulation model, while the blocking delay is an output parameter. Thus, the blocking delay is measured as a function of the communication delay.

For each simulation run, we assume the communication delay to have certain probability distribution, but we vary the probability distribution for different simulation runs. We use only hypo-exponential and hyper-exponential distributions; therefore each distribution can be characterized by its average and standard deviation.

Since the communication delay, consisting of communication network delay, IO delay, and CPU delay, is an input parameter and is modeled by an abstract probability distribution function, we make no assumptions about the characteristics of the underlying communication network, IO devices, and CPUs, and their relative performances. In fact, communication networks, IO devices, and CPUs of various performance characteristics are modeled by different probability distribution functions. For example, a high bandwidth and lightly loaded system has small variation in communication delay, thus it can be modeled by a distribution function with small standard deviation, while a low bandwidth and heavily loaded system can be modeled by a distribution function with large standard deviation. Notice that the average communication delay is used as the simulation time unit; therefore the average communication delay is not a factor in the simulation model. Thus the simulation results, especially the blocking delay, must be scaled according to the actual average communication delay.

Besides communication delay, input parameters of the simulation model include average transaction size, multiprogramming level, database size, and ratio of read-only transactions to update-only transactions entering the system. Besides blocking delay, performance measures (output parameters) include the probability of conflict of lock requests, the probability of deadlock and restart of lock requests, and the number of locks held by a transaction when the transaction deadlocks and restarts.

We did not explicitly include the frequency of local and remote data requests as an input parameter because it is captured by the probabilistic distribution of the communication delay. For example, a system with mostly local data requests can be modelled by a distribution function with small mean value. Neither did we include locking granularity as an input parameter because locking granularity is a function of the database size and the transaction size; increasing the granularity is equivalent to decreasing the database size and the transaction size. Moreover, we simulated only random access to the database -- every locking granule has the same probability of being accessed by a lock request -- because our previous study (Lin[2]) showed that more concentrated access distributions had the same results as the random access distribution in heavier load. The input parameters of the simulation model are discussed further in the remainder of the section.

For a database size (DZ) of N, N locks and N queues for the locks are simulated. Deadlock can occur, and the transaction in the deadlock cycle that holds the least number of locks aborts and restarts immediately. The reason we choose this particular transaction to abort is that our previous study (Lin[3]) concludes that this deadlock resolution algorithm performs best in all system and application environments.

Transaction size (TZ) is assumed to be exponentially distributed. The average of the distribution varies among different simulation runs, but remains fixed within a simulation run. A transaction requests locks sequentially, but different transactions request locks asynchronously. This model is general enough to include all transaction types of interest. For example, to model transactions in which read requests and update requests respectively are issued in parallel only once, the transaction size can be set to two.

After a lock request is granted, a transaction waits for a period of time before requesting another lock. The period of time is the communication delay discussed previously. The average of the communication delay is fixed at one for all simulation runs, but the standard deviation varies among different simulation runs. The simulation results can easily be scaled to whatever the actual average of the communication delay may be.

For a simulation run, the multiprogramming level (MP) is fixed; thus the model is closed, and a new transaction is generated and started only after one completes. The results of the simulation are presented in the next section.

3.3 Simulation Results

We simulated three different distributions of communication delay. All are erlangian and have the same average delay of one time unit; one has a standard deviation of 0.368, the second 1.87, and the third 5.28. For each standard deviation of communication delay, we simulated three different multiprogramming levels, three average transaction sizes, and three database sizes, for a total of 81 system configurations. Figures 3.1 through 3.8 show the results.

From the results we can conclude that the standard deviation of communication delay has no effect on the probability of conflict and the probability of deadlock of a lock request, or on the number of locks held when a transaction deadlocks. But it does have an effect on the average waiting time of blocked lock requests (blocking delay) — the larger the standard deviation, the longer the average waiting. We discuss the details of these observations in the rest of the section.

Each point of Figure 3.1 represents the probabilities of conflict (PC) of two system configurations with same multiprogramming level, average transaction size, and database size, but with different standard deviation of communication delay (DEV). One has a standard deviation of 0.368, and the other of 5.28. The X-coordinate of the point represents the probability of conflict of the former configuration, while the Y-coordinate represents the probability of conflict of the latter configuration. From the figure, we can see that all points lie very close to the diagonal line — implying that two system configurations with widely different standard deviations of communication delay have the same probability of conflict. Thus we can conclude that the standard deviation of communication delay has no effect on the probability of a lock request conflicting with another lock request. The reason is that the probability of conflict of a lock request depends only on the total number of locks outstanding in the system, which our simulation results show to be independent of the standard deviation of communication delay.

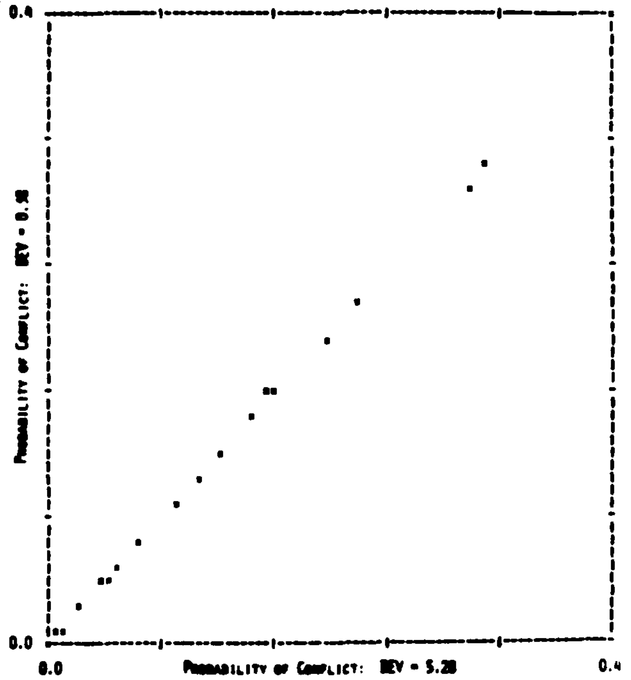


Figure 3.1 PC (DEV=5.28) Vs PC (DEV=.368)

Similar to Figure 3.1, Figure 3.2 represents the probability of deadlock and abortion of a lock request for the two different standard deviations of communication delay. From the figure, we can also con-

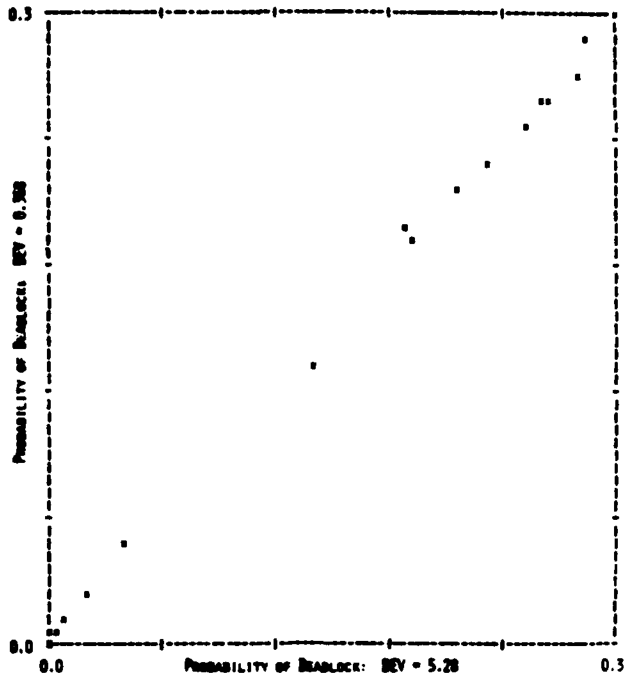


Figure 3.2 PD (DEV=5.28) Vs PD (DEV=.368)

clude that the standard deviation of communication delay has no effect on the probability of deadlock and abortion of a lock request.

The reason is that the probability of deadlock of a lock request depends on the total number of locks outstanding in the system, and on how these locks are distributed among transactions. Our simulation results show that these two factors are independent of the standard deviation of communication delay.

Figures 3.3a through 3.3c plot, for average transaction size of 4, 16, and 32 respectively, the average waiting delay of blocked lock requests against the standard deviation of communication delay for various system configurations. They show that the average waiting delay increases linearly with the standard deviation of communication delay when the average transaction size (TZ) and the multiprogramming level (MP) are small. Otherwise the increase is less than linear with communication delay variation. In fact, when the average transaction is larger than 32, the variation has little effect on the waiting delay. This can be explained by the following example. If all the transactions in the system are small, and the standard deviation of communication delay is large, then the time required by each transaction to complete varies greatly, say from 10 time units to 35 time units, with an average

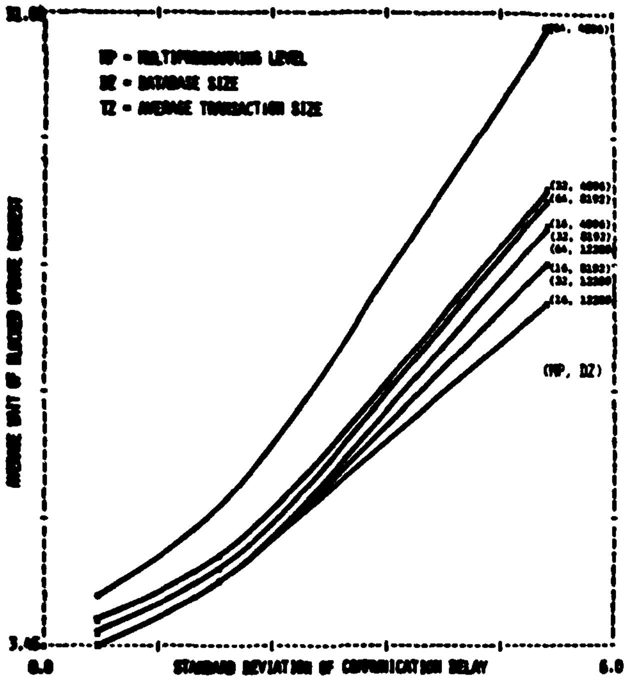


Figure 3.3a Average Blocking Delay when TZ=4

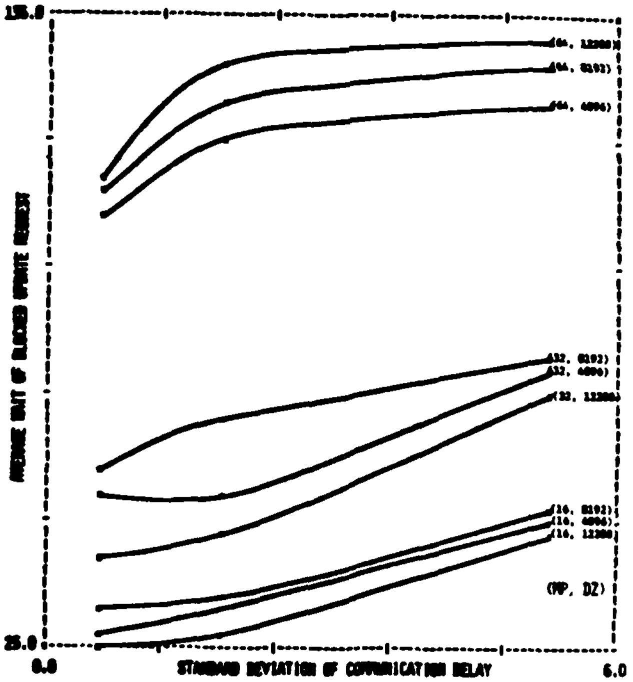


Figure 3.3b Average Blocking Delay when TZ=16

of 15 time units. Since blocked requests tend to wait for transactions that have been in the system longer, these blocking transactions tend to take from 15 to 35 time units to complete, with an average of 25 time units. But if the standard deviation of communication delay is small, then the time required by a transaction to complete varies less, say from 10 to 20 time units (with the same average of 15 time units). But the blocked requests most likely wait for transactions that require from

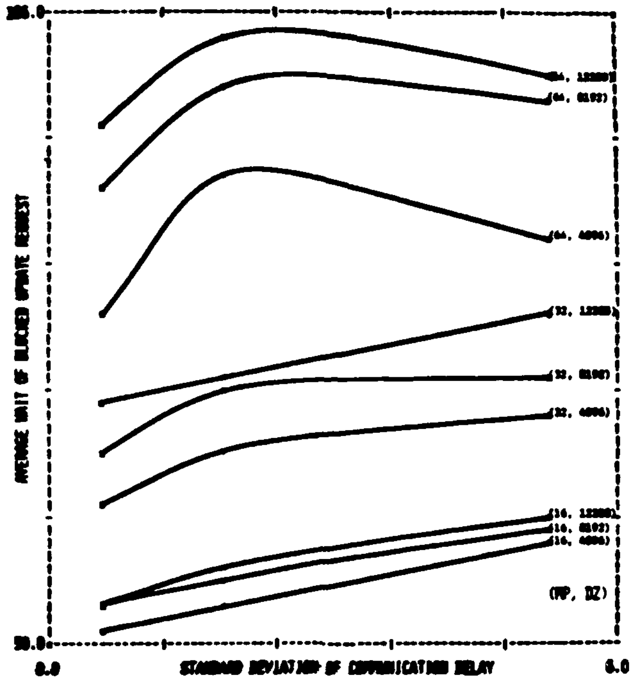


Figure 3.3c Average Blocking Delay when TZ=32

15 to 20 time units to complete, with an average of 17 time units, which is much smaller than the 25 time units of the previous case. Therefore the blocked requests in a configuration with larger standard deviation of communication delay tend to wait longer if the average transaction size is small.

If all transactions are large, then the time required by a transaction to complete varies little with the standard deviation of communication delay. The communication delay between two consecutive lock requests by the same transaction may vary widely if the standard deviation of communication delay is large, but since each transaction requests many locks, these variations eventually average out within each transaction. Therefore the average waiting delay of blocked requests is relatively invariant with the standard deviation of communication delay if the average transaction size is large.

In Figure 3.4, the X-coordinate and the Y-coordinate of each point represent respectively the average wait and the standard deviation of the wait of blocked lock requests of a system configuration. These 58 points represent the 81 runs with some runs overlapping on the same points. The figure shows that the standard deviation of the waiting delays of blocked requests is a fixed ratio of the average waiting delay regardless of multiprogramming level, average transaction size, database size, and communication delay variation. This observation implies that the waiting delay of blocked lock requests may have a fixed distribution function regardless of the system configurations; but the average of the distribution function is dependent on the system configurations as Figure 3.3a through 3.3c show. We plotted the distribution functions for some of the configurations, and all of them look similar to the one shown in Figure 3.3, which closely approximates 2-stage hypoexponential (Erlangian) distribution function. In fact, the fixed ratio of Figure 3.4 (the slope of the line) approximates the standard deviation of a 2-stage hypoexponential distribution.

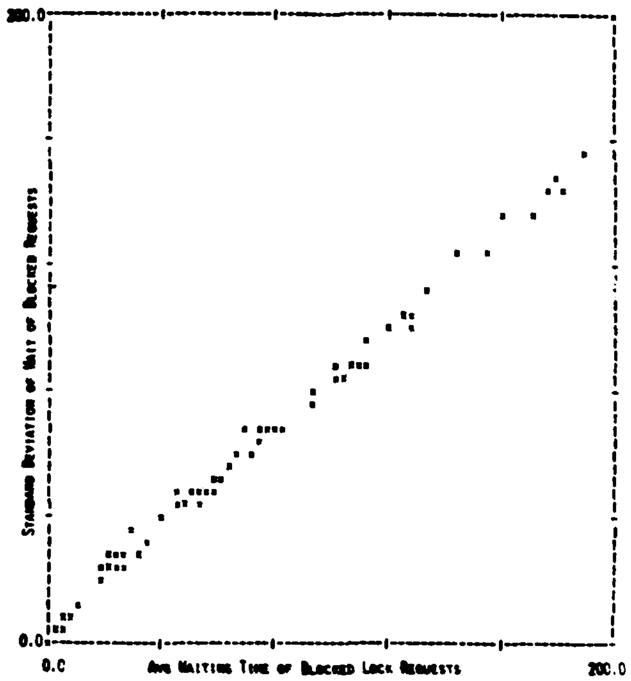


Figure 3.4 Average Vs Standard Deviation of Blocking Delay

For each system configuration, we compute the average number of locks held by transactions when they deadlock and abort, denoted by LOCKS_AT_DEADLOCK. Each point of Figure 3.6 represents the average LOCKS_AT_DEADLOCKs of two system configurations with same multiprogramming level, average transaction size, and database size, but with different standard deviation of communication delay; one has standard

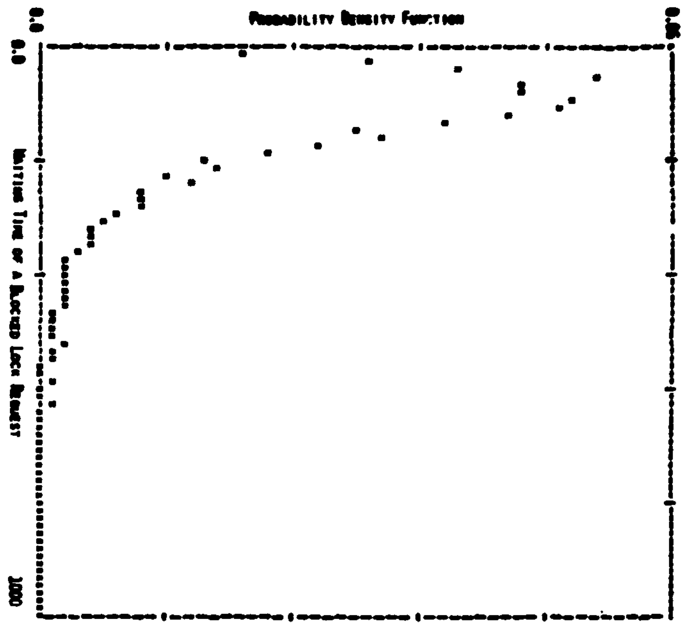


Figure 3.5 Distribution of Blocking Delay

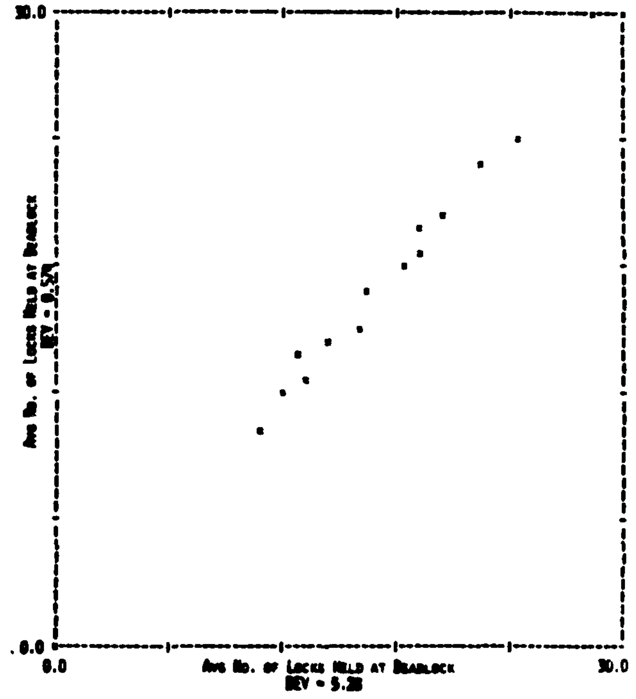


Figure 3.6 LOCKS_AT_DEADLOCK (DEV=5.28) Vs.
LOCKS_AT_DEADLOCK (DEV=.574)

deviation of 0.574, the other of 5.28. The X-coordinate represents the average LOCKS_AT_DEADLOCK of the former configuration, while the Y-coordinate represents the latter configuration. From the figure, we can see that all points lie very close to the diagonal line, implying that

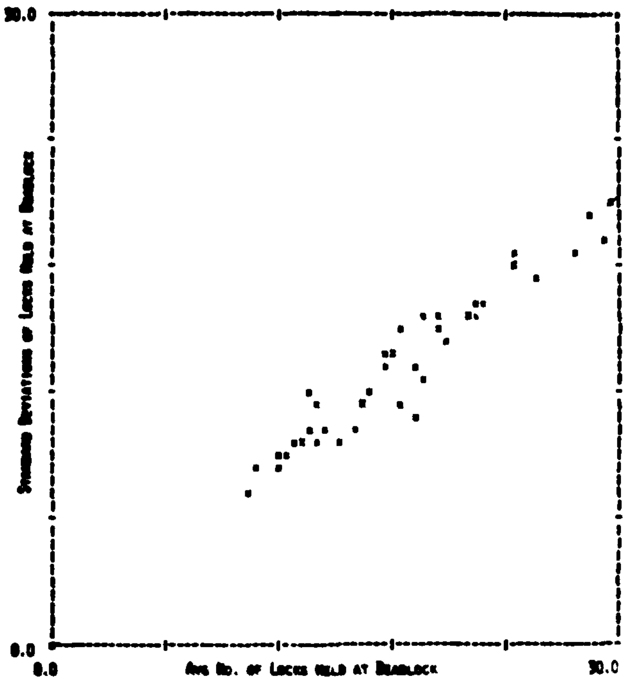


Figure 3.7 Average Vs Standard Deviation of LOCKS_AT_DEADLOCK

two configurations with widely different standard deviations of communication delay have the same LOCKS_AT_DEADLOCK. Thus we can conclude that the communication delay variation has no effect on the number of locks held by a transaction when the transaction deadlocks and aborts.

For each system configuration, the standard deviation of the number of locks held by transactions when they deadlock and abort, represented by DEV_LOCKS_AT_DEADLOCK is plotted against the LOCKS_AT_DEADLOCK in Figure 3.7. This plot approximates a straight line, indicating that DEV_LOCKS_AT_DEADLOCK may be a fixed ratio of LOCKS_AT_DEADLOCK. Regression analysis indicates the ratio is about 0.70, implying that the number of locks held at deadlock may have a 2-order negative binomial distribution. This distribution is obtained by throwing a biased coin repeatedly until we obtain the second success. The number of throws represents the number of locks held by a transaction when the transaction deadlocks and aborts. The mean of the distribution function depends on the bias of the coin, and the bias of the coin depends on the multiprogramming level, the average transaction size, and the database size. We plotted the distribution functions of a few system configurations and found all of them looking like the one shown in Figure 3.8, which closely approximates a 2-order negative binomial distribution.

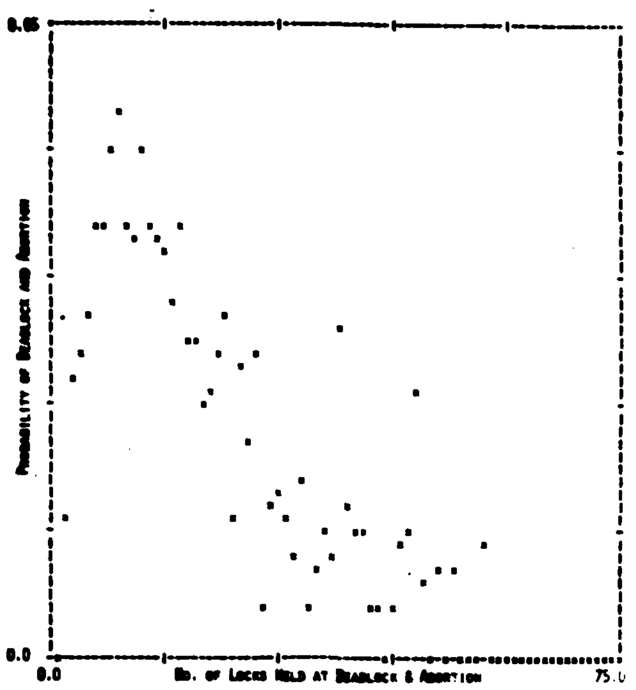


Figure 3.8 Distribution of LOCKS_AT_DEADLOCK

3.4 Conclusion

In summary, we can conclude that communication delay variation has no effect on the chance of a lock request conflicting or deadlocking with another lock request. It also has no effect on the number of locks held by a transaction when the transaction deadlocks and aborts. In fact, the distribution function of the number of locks held by a transaction when it deadlocks and aborts has a 2-order negative binomial distribution with its average and standard deviation independent of the communication delay variation.

The blocking delay of a blocked lock request has a 2-stage Erlangian distribution regardless of the standard deviation of communication delay. The mean of the distribution is also independent of the standard deviation of communication delay, if the average transaction size is large. However, if the average transaction size is small, the mean of the distribution function depends on the standard deviation of communication delay -- the larger the variation, the longer the average blocking delay. But in many cases, conflict occurs rarely. Therefore its effect on system performance is insignificant.

These results are important to performance modeling of distributed concurrency control, because they eliminate the standard deviation of communication delay as one of the system parameters that affect system performance. For an analytical model, this means that the communication delay can be assumed to have an exponential distribution which simplifies the model. For a simulation model, this means geometric reduction of the number of simulation runs.

3.5 References

Bad[1] Badal, D.Z., et al., "A proposal for Distributed Concurrency Control for Partially Redundant Distributed Database System," 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, 1978

Ber[1] Bernstein, P., Goodman, N., "Concurrency Control in Distributed Database Systems", ACM Computing Survey, Vol. 13, No. 2, June 1981

Ell[1] Ellis, C.A., "A Robust Algorithm for Updating Duplicate Databases," 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977

Gar[1] Garcia-Molina, H., "Performance of Update Algorithms For Replicated Data in a Distributed Database", Ph.D. Thesis, Dept. of Computer Science, Stanford University, June 1979

Lin[1] Lin, W.K., "Performance Evaluation of Two Concurrency Controls Mechanisms in a Distributed Database System," Sigmod-81 Int'l Conf. on Management of Data, Apr. 1981, Ann Arbor, MI

Lin[2] Lin, W.K., Nolte, J., "Performance of Two Phase Locking," 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Feb. 16-19, 1982, Pacific Grove, CA.

Lin[3] Lin, W.K., et al., "Distributed Database Control & Allocation: Semi-Annual Report", Technical Report, Computer Corporation of America, Cambridge, MA.

Lin[4] Lin, W.K., "Concurrency Control In a Multiple Copy Distributed Database System," 4th Berkeley Workshop on Distributed Data Management and Computer Networks, Aug. 1979

Rie[1] Ries, D., "The Effect of Concurrency Control on Database Management System Performance", Ph.D. Thesis, Electronics Research Lab, Univ of Cal., Berkeley, 1979

Ros[1] Rosenkrantz, D.J., et al., "System Level Concurrency Control for Distributed Database Systems," ACM Trans. on Database Systems, Vol 3, No. 2, June 1978

Ste[1] Sterns, R.E., Rosenkrantz, D.J., et al., "Distributed Database Concurrency Controls Using Before-Values," Sigmod-81 Int'l Conf. on Management of Data, Apr. 1981, Ann Arbor, MI

Sto[1] Stonebraker, M., et al., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," IEEE Trans. on Software Engineering, Vol SE-5, No. 3, May 1979

Tha[1] Thanos, C., et al., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed Database System," Lecture

Notes in Computer Science, ed. G. Goo & J. Hartmanis, Springer-Verlag, NY, 1981

Tho[1] Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Database," ACM Trans. on Database Systems, Vol 4, No. 2, June 1979

SECTION IV

READ ONLY TRANSACTIONS AND TWO PHASE LOCKING*

Wente K. Lin
Jerry Nolte

* This paper appeared in the Second IEEE Symposium on Reliability in Distributed Software and Database Systems (July 1982).

4. Read Only Transactions and Two Phase Locking

Abstract

Intuition tells us that in a distributed DBMS using two phase locking, the ratio (denoted by R/W) of read-only to update transactions affects system performance -- the higher the ratio, the better the performance. Read-only transactions only request share locks, and thus should cause fewer conflicts and deadlocks among all transactions. Therefore both read-only and update transactions are expected to perform better if R/W is higher. This paper reports the results of a study contradicting this intuition, and discusses the relationship between the R/W ratio and system performance in detail.

4.1 Introduction

Many distributed concurrency control algorithms have been proposed (Bad[1], Ber[1], Ell[1], Gar[1], Lin[4], Ros[1], Ste[1], Sto[1], Tho[1]). But how well do they perform? A few researchers have attempted to compare the performance of different algorithms (Gar[1], Lin[1], Lin[2], Mun[1], Rie[1], Tha[1]). Unfortunately they all used different assumptions about system and application parameters. Furthermore, they compared different algorithms using different performance measures. For example, in Lin[1], two timestamping algorithms are compared, and the performance measure used is the average response time.

In Rie[1], two two-phase locking algorithms — the centralized method and the primary copy method — are compared. Performance measures include utilization of devices, average transaction response time, and others. In addition, a transaction must obtain all its locks before it can start, no duplicate data is allowed, and multiprogramming level is assumed to have no effect on system performance.

In Gar[1], three two-phase locking algorithms are compared — the centralized method, the voting method, and the ring method. Both utilization of devices and average response time are performance measures. A few assumptions are made: all transactions are update transactions; multiprogramming level is one at each node; a transaction must obtain all locks before it proceeds; a transaction requests all locks in parallel; and the database is fully duplicated in every site (performance of partitioned database is treated briefly).

In Lin[2], distributed two-phase locking algorithms are abstracted into one model, and the relation between system blocking behavior (conflict and deadlock) and various system and application parameters is studied. Two access distributions of the database are simulated. The first one has a uniform distribution: every data granule is equally likely to accessed by a lock request; the second one has 20% of the database accessed by 80% of the lock requests. It concludes that the more concentrated database access distribution has the same effect on system blocking behavior as the uniform distribution in heavier load.

In Tha[1], two two-phase locking algorithms are simulated — basic two-phase and centralized two-phase. The performance measure used is the average transaction response time. The simulation model includes many system and application parameters, but results of only a few simulation runs with limited values of these parameters are presented.

These performance studies are very difficult to compare, and it is almost impossible to integrate their results. They compare different algorithms, and they use different assumptions about system and application environments and different measures for system performance. Therefore we began a major project in order to compare the principal distinct distributed concurrency control and reliability algorithms, using the same model, same assumptions, same performance (output) parameters, and the same system and application (input) parameters. This paper reports part of the findings of this project. In particular, this paper reports our findings about the relationship between read-only transactions and the performance of two phase locking. We found that, when the ratio of read-only transactions to update transactions increases from 1/3 to 3/1, the response times of both read-only and update transactions and total system through-put remain unchanged, except when the system load is extremely heavy and transactions are long.

This paper is organized as follows. Section 4.2 describes the simulation model, Section 4.3 discusses the simulation results, and Section 4.4 concludes the study.

4.2 The Simulation Model

Because this paper reports part of the findings of a larger project, we describe the simulation model of the larger project first. We model a transaction as a sequence of lock requests. A lock request incurs two kind of delays. The first, called blocking delay, occurs because of lock conflict (two requests ask for the same lock). The second delay, called communication delay, consists of communication network delay, IO delay, CPU processing delay, and others.

Blocking delay and communication delay are affected by many factors, in additional to the R/W ratio. These include average transaction size (number of locks requested by a transaction), multiprogramming level (number of transactions running concurrently), database size (number of data granules), access distribution of the database (probability of each locking granule being accessed by a lock request), communication network, IO devices, memory size, CPU speed, and others (locking granularity is not explicitly considered because it is a function of the transaction size and the database size). Thus to accurately evaluate these two delays, we must include all these factors in the simulation model, and this is too expensive to do directly.

To simplify the simulation, we divided the simulation into two steps. During the first step, we considered the communication delay as one of the input parameters, and the blocking delay as one of the output parameters (performance measures); thus the blocking delay is measured as a function of the communication delay. For each simulation run, we assumed the communication delay to have certain probability distribution, but we varied the probability distribution for different simulation runs. We used only hypo-exponential and hyper-exponential distributions; therefore each distribution can be characterized by its average and standard deviation.

Since the communication delay, consisting of communication network delay, IO delay, and CPU delay, is an input parameter and is modeled by an abstract probability distribution function, we made no assumptions about the characteristics of the underlying communication network, IO devices, and CPU, and their relative performances. In fact, communication networks and IO devices with different performance characteristics are modeled by different distribution functions. For example, a distribution function with small standard deviation simulates a high bandwidth or a lightly loaded system, while a distribution function with large standard deviation simulates a low bandwidth or a heavily loaded system.

Besides communication delay, system and application parameters (input parameters) include average transaction size, multiprogramming level, database size, ratio of read-only transactions to update-only transactions, and access distribution to the database. Besides blocking delay, performance measures (output parameters) include probability of conflict and deadlock among lock requests, average response times of read-only and update lock requests, and system through-put.

During the second step of the simulation, the performance measures obtained during the first step will be used to avoid simulating the management of locks and timestamps. For example, when a two-phase locking algorithm is simulated in the second step, the probability functions of conflict and deadlock obtained during the first step will be used in conjunction with a random number generator to decide whether a lock request must conflict and deadlock. No locks and queues of lock requests will be simulated explicitly. The second step simulation is being continued and will be presented in a future report. The first step simulation model is described in the rest of this section.

For a database size (DZ) of N, N locks and N queues for the locks are simulated. Deadlock can occur, and the transaction in the deadlock cycle that holds the least number of locks aborts and restarts immediately. The reason we choose this particular transaction to abort is that our previous study (Lin[3]) concludes that this deadlock resolution algorithm performs best in all system and application environments we have simulated.

Transaction size (TZ) is assumed to be exponentially distributed. The average of the distribution varies among different simulation runs, but remains fixed within a simulation run. A transaction requests locks sequentially, but different transactions request locks asynchronously. A transaction model in which read locks and write locks respectively are requested in parallel is thus equivalent to our transaction model with transaction size equal to two.

After being granted a lock request, a transaction waits for a period of time before requesting another lock. The period of time is the communication delay discussed previously. The average of the communication delay is fixed at one for all simulation runs, but the standard deviation varies among different simulation runs. The simulation results can easily be scaled to whatever the actual average of the communication delay may be.

The multiprogramming level (MP) is fixed within a simulation run, but varies among different simulation runs; thus the model is closed: a new transaction is generated and started only after one completes.

The access distribution to the database is random: every lockable unit has the same probability of being accessed by a lock request. We use this uniform distribution because our previous study (Lin[2]) shows that more concentrated distributions have the same results as the uniform distribution in heavier load.

Part of the findings concerning the relationship between the R/W ratio and system performance of the first step simulation is presented in the next section.

4.3 Simulation Results

We ran the simulation program many times with different values of multiprogramming level, average transaction size, database size, standard deviation of communication delay, and R/W ratio. The table below shows the input parameters and their values used in the simulation.

Input Parameter	Values Used
transaction size (TZ)	4,16,32
multiprogramming level (MP)	16,32,64
database size (DZ)	4096,8192
R/W ratio	1/3,1/1,3/1
standard deviation of communication	0.573, 0.75,1.87,5.28

Figure 4.0

We present only the results of the standard deviation of communication delay of 0.75, because we found that the relationship between the R/W ratio and the performance of the two phase locking does not change with different standard deviation of communication delay (Lin[3]). Tables 1 through 9 show the results, which are also rearranged and plotted in Figures 4.1 through 4.7.

In Figure 4.1 each point represents the probabilities of conflict of update lock requests for two system configurations with same MP, TZ, DZ, and DV, but with different R/W. Different points of the figure represent different configurations with different MP, TZ, DZ, or DV. The X-coordinate represents the probability of conflict for the system with R/W equal to 1/3, and the Y-coordinate the system with R/W equal to 3/1. The points in the figure lie close to the diagonal line, implying that the probabilities of conflict of update requests are the same for each two configurations with different R/W ratio. The R/W ratio has no effect on the probability of conflict of update lock requests.

Figure 4.2 plots a similar graph for the probability of deadlock of update lock requests, and the points also lie close to the diagonal line. Thus the R/W ratio also has little effect on the probability of deadlock of update lock requests.

These two observations contradict the intuition that higher R/W ratio reduces conflict and deadlock for update transactions because of more share locks and less exclusive locks in the system. To gain more insight about this unexpected result, during each simulation run we examined the locks outstanding in the system. We found that even though the ratio of share locks to exclusive locks did increase with higher R/W ratio, on the average the total number of locks outstanding in the system at any time varies little with the R/W ratio. And the total number of locks outstanding in the system determines the probability of conflict and the probability of deadlock of update requests, because update requests conflict and deadlock with both read and update lock requests.

Figure 4.3 plots a similar graph for the average blocking delay of blocked update lock requests. The figure shows that the R/W ratio does have an effect, though only a small one, on the blocking delay of blocked update lock requests. Specifically, the average waiting decreases a little when the R/W increases from 1/3 to 3/1. Our results (Table .8) show that with a higher R/W, read-only transactions complete slightly faster; thus update transactions wait slightly shorter for blocking read-only transactions.

We have examined the effect of R/W ratio on the probability of conflict and deadlock of update lock requests. Here, we examine how that effect translates into system performance in terms of response time. Table 4 shows the average response time of completed update lock requests. The average response time includes time wasted due to transaction abortion. Notice that if there is no blocking delay and transaction abortion, then the average response time of update lock requests must be one. The table shows that the only acceptable response times occur when the average transaction size is four or the load is less than 1%. The load is defined as the product of multiprogramming level and the average transaction size divided by the database size. The table shows that within the acceptable range of transaction size and system load, the average response time of update lock requests varies little with the R/W ratio. This is expected because of the invariance of the probability of conflict, probability of deadlock, and blocking delay of update requests with respect to R/W ratio.

We next examine the effect of the R/W ratio on read-only lock requests. Similar to Figures 4.1 and 4.2, Figure 4.4 and Figure 4.5 plot the probability of conflict and the probability of deadlock of read-only lock requests respectively. These figures show that higher R/W ratios reduce significantly both the probability of conflict and the probability of deadlock of read-only lock requests.

Similar to Figure 4.3, Figure 4.6 plots the average blocking delay of blocked read-only lock requests, and the figure shows that the R/W ratio has little effect on the blocking delay of read-only requests. This occurs because blocked read requests wait only for update transactions, and we have shown previously that the R/W ratio has little effect on the response time of update transactions.

We have shown that a higher R/W ratio reduces the probability of conflict and deadlock of read-only lock requests, but it has no effect on their blocking delay. What does this mean in terms of the average response time of read-only lock requests? Table 8 shows the average response time of read-only requests, indicating that acceptable response times occur when the average transaction size is 4 or the system load is less than 1%. Within this range of transaction size and system load, when the R/W increases from 1/3 to 3/1, the average response time of read-only requests decreases only slightly.

This result is surprising because we have previously observed that the probability of conflict and deadlock of read-only requests decreases significantly when the R/W ratio increases from 1/3 to 3/1. But because the probability of conflict and deadlock is very small to begin with when the transaction size and system load are within acceptable range, the reduction in the probability of conflict and deadlock does not improve significantly the response time of read-only requests.

We next examine the relationship between the R/W ratio and system through-put. The results are shown in Table 9. The table shows that, within the acceptable range of transaction size and system load, the system through-put does not increase significantly, when the R/W ratio increases from 1/3 to 3/1,

To gain more insight, we did some time series analysis and found that regardless of the R/W ratio in the incoming transaction stream, the system is eventually saturated with mostly update transactions. Figure 4.7 shows the time series of numbers of update transactions active in a system with R/W ratio equal to 3/1 and multiprogramming level equal to 64. The figure shows that the number of update transactions active in the system is stablized at about 95% of the multiprogramming level, even though 75% of incoming transactions are read-only transactions. This explains why the system does not perform much better when the R/W ratio increases from 1/3 to 3/1, because the system is clogged up with update transactions that complete slowly.

You might have noticed that the database sizes used are relatively small compared to actual databases. But we have pointed out that the results apply to systems with short transactions and moderate loads; therefore the results apply to systems with larger database size, because the larger the DZ, the lighter the load.

4.4 Summary

We simulated two phase locking in various system and application environments. We found that R/W has little or no effect on the probability of conflict and deadlock and the blocking delay of update lock requests. In addition, the R/W ratio has little effect on the response times of update lock requests.

We also found the R/W ratio has little effect on the blocking delay of read-only transactions. However, we found that the R/W ratio has significant effect on the probabilities of conflict and deadlock of read-only transactions. Increase in R/W ratio significantly reduces the percentage of probabilities of conflict and deadlock of read-only transactions. But if the average transaction size is small or the system load is light, then this reduction in the probability of conflict and deadlock reduces only slightly the response time of read-only lock requests. And the overall system through-put is little effected by the R/W ratio.

4.5 References

Bad[1] Badal, D.Z., et al., "A Proposal for Distributed Concurrency Control for Partially Redundant Distributed Database System," 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, 1978.

Ber[1] Bernstein, P., N. Goodman, "Concurrency Control in Distributed Database Systems", ACM Computing Surveys, Vol. 13, No. 2, June 1981.

Ell[1] Ellis, C.A., "A Robust Algorithm for Updating Duplicate Databases," 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.

Gar[1] Garcia-Molina, H., "Performance of Update Algorithms For Replicated Data in a Distributed Database", Ph.D. Thesis, Dept. of Computer Science, Stanford University, June 1979.

Lin[1] Lin, W.K., "Performance Evaluation of Two Concurrency Controls Mechanisms in a Distributed Database System", Sigmod-81 International Conference on Management of Data, April 1981, Ann Arbor, MI.

Lin[2] Lin, W.K., J. Nolte, "Performance of Two Phase Locking", 6th Berkeley Workshop on Distribute Data Management and Computer Networks, Feb. 16-19, 1982, Pacific Grove, CA.

Lin[3] Lin, W.K., et al., "Distributed Database Control & Allocation: Semi-Annual Report", Technical Report, Computer Corporation of America, Cambridge, MA.

Lin[4] Lin, W.K., "Concurrency Control In a Multiple Copy Distributed Database System," 4th Berkeley Workshop on Distributed Data Management and Computer Networks, Aug. 1979.

Rie[1] Ries, D., "The Effect of Concurrency Control on Database Management System Performance", Ph.D. Thesis, Electronics Research Lab, Univ of Cal., Berkeley, 1979.

Ros[1] Rosenkrantz, D.J., et al., "System Level Concurrency Control for Distributed Database Systems," ACM Trans. on Database Systems, Vol. 3, No. 2, June 1978.

Ste[1] Sterns, R.E., D.J. Rosenkrantz, et al., "Distributed Database Concurrency Controls Using Before-Values," Sigmod-81 International Conference on Management of Data, April 1981, Ann Arbor, MI.

Sto[1] Stonebraker, M., et al., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," IEEE Trans. on Software Engineering, Vol. SE-5, No. 3, May 1979.

Tha[1] Thanos, C., et al., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed Database System," Lecture Notes in Computer Science, editor G. Goo & J. Hartmanis, Springer-Verlag, NY, 1981.

Tho[1] Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Database," ACM Trans. On Database Systems, Vol. 4, No. 2, June 1979.

SECTION V

Basic Timestamp, Multiple Version Timestamp, and Two Phase Locking*

Wente K. Lin

Jerry Nolte

* A version of this paper appeared in the Third Semi-annual technical report of the DDB Control and Allocation Project.

5. Basic Timestamp, Multiple Version Timestamp, and Two Phase Locking

Abstract

Using simulation, we compare the performance of the basic timestamp, the multiple-version timestamp, and the two phase locking concurrency control protocols. We find that in every system configuration we have simulated the multiple version timestamp protocol performs only marginally better than the basic timestamp protocol. In addition, we find that when the average transaction size is small, both timestamp protocols outperform the two phase locking protocol. But when the average transaction size is large, the two phase locking protocol outperforms both timestamp protocols.

5.1 Introduction

Many distributed concurrency control algorithms have been proposed (Bad[1], Ber[1], Ell[1], Gar[1], Lin[3], Ros[1], Ste[1], Sto[1], Tho[1]). But how well do they perform?

A few researchers have attempted to compare performance of different algorithms (Gar[1], Lin[1], LN[1], LN[2], LN[3], Mun[1], Rie[1], Tha[1]), and only one of them studied the performance of timestamp protocols (Lin[1]).

These performance studies are very difficult to compare, and it is almost impossible to integrate their results. They compare different algorithms, and they make different assumptions about system and application environments and employ different measures for system performance. Therefore we began a major project that compared the principal distinct distributed concurrency control and reliability algorithms, using the same model, assumptions, performance (output) parameters, and system and application (input) parameters. Some results of the project concerning the two phase locking have been reported in Lin[2], LN[1], LN[2], and LN[3]. This paper reports some of the results of this project that concern timestamp protocols.

In particular, this paper reports our findings about the performance of the basic timestamp and the multiple version timestamp protocols (Ber[1]), and about the comparison of their performance to the performance of the two phase locking protocol. We found that, contrary to our intuition, the multiple version timestamp protocol did not significantly increase the throughput of read-only transactions over the basic timestamp protocol; neither did it improve the throughput of update transactions. We also found that both timestamp protocols performed much better than the two phase locking protocol when the average transaction size was small. But when the average transaction size was large, the two phase locking protocol outperformed both timestamp protocols.

This paper is organized as follows. Section 5.2 describes the overall simulation model. Section 5.3 describes the specifics of the basic timestamp and the multiple version timestamp models. In addition, the simulation results for these two models are discussed. Section 5.4

discusses the simulation results of a modified model in which the ratio of read-only transactions to update transactions is fixed inside the system, instead of in the incoming transaction stream. Section 5.5 compares the two phase locking with the basic timestamp protocol, and Section 5.6 concludes the study. Section 5.7 contains the references.

5.2 The Simulation Model

Performance of a concurrency control algorithm in a DBMS depends on many aspects of the entire system. These include system characteristics such as multiprogramming level (number of transactions running concurrently), database size (number of data granules) and granularity with which data can be locked or accessed, communication network, IO devices, memory size, CPU speed, number of nodes in the system, and distribution of the data among these nodes. Performance is also affected by the nature of the application -- the transactions executed to read or update the database. Transaction characteristics include transaction size (number of data granules requested by each transaction), the frequency of local and remote requests for data, access distribution of the database (probability of each data granule being accessed by a data request), and whether the transactions only read data or update the database. Thus, to accurately evaluate the performance of a concurrency control algorithm, we must include all these factors in the simulation model, and this is too expensive to do directly.

To simplify the simulation, we model the system and transactions in a highly functional model. Much of the detail of a real distributed system is captured in a few parameters that are used as inputs to the simulation model. This approach permits us to greatly reduce the number of simulation runs necessary and the complexity of the model, while retaining most of the impact that these details have on the performance of the concurrency control algorithms.

We model a transaction as a sequence of data requests, each requesting a data granule. The size of the data granule is irrelevant in our model. Between two consecutive data requests, a transaction incurs a delay called processing delay. The processing delay consists of communication network delay, IO delay, and CPU processing delay. Communication network, IO devices, and CPUs are not simulated in detail in our model. Instead we use the processing delay as an input parameter to our simulation model. For each simulation run, we assume the processing delay to have a certain probability distribution, but we vary the probability distribution for different simulation runs. We use only hypoexponential and hyperexponential distributions; therefore each distribution can be characterized by its average and standard deviation.

Since the processing delay (consisting of communication network delay, IO delay, and CPU delay), is an input parameter and is modeled by an abstract probability distribution function, we make no assumptions about the characteristics of the underlying communication network, IO devices, and CPUs, and their relative performances. In fact, communication networks and IO devices that have different performance characteristics are modeled by different distribution functions. For example, a distribution function that has small standard deviation models a high bandwidth or a lightly loaded system, while a distribution function that has large standard deviation models a low bandwidth or a heavily loaded system.

Besides processing delay, input parameters of the simulation model include average transaction size (TZ), multiprogramming level (MP), database size (DZ), ratio of read-only transactions to update-only transactions (R/W) entering the system, and access distribution to the database.

We did not explicitly include the frequency of local and remote data requests as an input parameter because it is captured by the probability distribution of the processing delay. For example, a system that executes mostly local data requests can be modelled by a distribution that has a small mean value. Neither did we include the data granularity as an input parameter, because the data granularity is a function of the database size and the transaction size. Increasing the granularity is equivalent to decreasing the database size and the transaction size. Moreover, we simulated only random access to the database. Every data granule has the same probability of being accessed by a data request. We used this uniform distribution because our previous study (LN[1]) showed that more concentrated distributions had the same results as the uniform distribution in heavier load. The details of these input parameters follow.

We simulated two kinds of transactions, read-only transactions and update transactions, and the R/W ratio determines their ratio in the incoming transaction stream fed to the simulation system.

Transaction size is assumed to be exponentially distributed with mean TZ. The mean TZ varies among different simulation runs, but remains fixed within a simulation run. We model a read-only transaction as a sequence of read requests, and an update transaction as a sequence of read requests (with intention to update later), followed by parallel update requests (each requesting only one data granule). Thus in an update request, we require that each data granule be read before it is updated. We assume that an update transaction has two phases: a read phase and a write phase. During the read phase, an update transaction issues a sequence of read requests, and during the write phase all updates are committed in parallel into the databases. This model is general enough to include all transaction types of interest. For example, to model a pure update transaction (one that does not read from the database) there will be only the write phase. To model transactions in which read requests are issued in parallel only once, the read phase will issue only one read request.

After being granted a data granule, a transaction waits for a period of time before requesting another granule. This period of time is the processing delay discussed previously. The average of the processing delay is fixed at one for all simulation runs, but the standard deviation varies among different simulation runs. The simulation results can easily be scaled to whatever the actual average of the processing delay may be.

The multiprogramming level (MP) is fixed within a simulation run, but it varies among different simulation runs; thus the model is closed and a new transaction is generated and started only after one completes.

Performance measures (output parameters) of the simulation model include probability of restart, system throughput, and others that will be mentioned when specific models are discussed.

Part of the results of the performance of the basic timestamp and multiple version timestamp protocols are presented in the following sections.

5.3 Basic Timestamp vs Multiple-Version Timestamp

In this section, we first describe the specifics of the basic timestamp and the multiple version timestamp models, and then we discuss the simulation results. Much of the detail of the protocol can also be found in Ber[1]. We describe the basic timestamp model first.

We assign a unique timestamp (drawn from the system clock) to each transaction when we initiate the transaction. We keep a read timestamp and a write timestamp with each data granule of the database. The read timestamp and the write timestamp record the timestamps of the last transactions, reading and writing respectively the data granule.

To synchronize an update transaction, during the read phase the timestamp of the update transaction is compared with the read and write timestamps of each data granule read. If the timestamp of the update transaction is smaller than the read timestamp, the update transaction is restarted immediately to avoid aborting it later when it tries to commit (since we never abort read-only transactions). If the timestamp of the update transaction is smaller than the write timestamp, then the update transaction is also restarted because it tries to read the data granule after a transaction that has a greater timestamp has updated the data granule. If the timestamp of the update transaction is larger than both read and write timestamps of the data granule, then it replaces the read timestamp of the data granule and the update transaction continues. During the write phase, the timestamp of the update transaction is again compared to the read and write timestamps of each data granule updated. If the timestamp of the update transaction is smaller than the read timestamp, the update transaction is again restarted. But if the timestamp of the update transaction is smaller than the write timestamp, the write operation is ignored. If the timestamp of the update transaction is larger than both the read and write timestamps of the data granule, the write timestamp of the data granule is replaced by the timestamp of the update transaction. If T(t) represents the timestamp of transaction t, and R(x) and W(x) the read timestamp and write timestamp of data granule x, the protocol can be summarized as follows. During the read phase of an update transaction t,

```
for each x read by t,
 if T(t)<R(x)            --> restart t;
 if T(t)<W(x)              —> restart t;
 if T(t)>R(x) & T(t)>W(x) —> replace R by T, read proceeds.
```

And during the write phase of an update transaction t,

```
 if T(t)<R(x) for any x updated by t   —> restart t;
 else for each x updated by t,
 if T(t)<W(x) —> update to x is ignored,
 if T(t)>W(x) —> replace W(x) by T(t), commit the update.
```

To process a read request from a read-only transaction, we compare its timestamp with the write timestamp of the data granule. If the write timestamp of the data granule is larger, then the read-only transaction is restarted; otherwise the read-only transaction continues, and the read timestamp of the data granule is replaced by the timestamp of the read request if the latter timestamp is greater than the former. In summary,

```
  T<W  —> restart
  T>W  —> read-only proceeds, and replace R by T if R<T.
```

Performance measures of the model include system throughput (number of requests completed per time unit) and the probability of restart for both read requests of read-only transactions and read requests of update transactions during the read phase. Since in the case of timestamping protocols, an update transaction may progress to the write phase and then conflict and abort, we also include the probability of restart of transactions (not data requests) during the write phase.

The multiple version timestamp model is very similar to the basic timestamp model. System and application parameters, and conflicts between data requests and data timestamps, were dealt with in the same way. However, in the multiple version model, we kept four read and four write timestamps for each data granule; the first one is the smallest and the fourth one the largest. However, because we did not simulate computation within each transaction, we did not keep the data values corresponding to the four write timestamps for each data granule. A read-only transaction can access earlier versions of the data if the timestamp of the read-only transaction is smaller than the largest write timestamp of the data granule to be accessed. But because we require an update transaction to read first what it writes, an update transaction can only read the latest version; if the R/W ratio is zero, this model degenerates to the basic timestamp model. For this reason, we did not simulate any system configuration with R/W equal to 0.

Since the probability of restart for read-only transactions is already small in the single version basic timestamping protocol, and since the number of versions does not affect the probability of restart for update transactions, we decided not to vary the number of versions. We compare the simulation results of both the basic and the multiple-version timestamp protocols in the following.

We first examine the probability that a read request (both read-only request and request of update transaction during the read phase) will conflict, resulting in the restart of its transaction. Figure 5.1 and Figure 5.2 respectively show these probabilities for the basic and the multiple-version timestamp protocols. We note that because read-only transactions never restarted in the multiple-version timestamp model, Figure 5.2 contains only data for update transactions during the read phase. We note also that, for some of the heavy load cases, the system thrashed and never stabilized; therefore the data are not reliable. However, they do qualitatively indicate what is happening. Comparing these two figures, we find very little difference between the basic timestamp and the multiple-version timestamp protocols in the probability of restart during the read phase.

We next examine the probability of restart of update transactions during the write phase. Figure 5.3 and Figure 5.4 show the results for the basic timestamp and the multiple-version timestamp protocols, and the difference between the two figures is very small.

For the basic and the multiple-version timestamp protocols, Figure 5.5 and Figure 5.6 show the system throughput, which is the number of completed (excluding those aborted) data requests per time unit. Notice that the average processing delay is always one and that there are always MP transactions running in the system; therefore if there is no transaction abortion, the throughput must equal MP, which is the maximum possible throughput. Combined read-only and update throughputs for system configurations that have average transaction size equal to 4 are within 10% of maximum possible. But combined throughputs of system configurations that have average transaction size (TZ) larger than 16 are less than 30% of the maximal throughput.

These two figures show system thrashing when the average transaction size is large or the system load is heavy. If the system is in equilibrium, write throughput should be very nearly 1/3 of the read throughput, since incoming transactions occur in that ratio. However, this is not true for TZ=32, or for TZ=16, MP=32 and 64. In these cases, the system thrashed and was jammed with long update transactions that never finished. These observations show that both timestamp protocols perform extremely poorly during long transactions or while bearing heavy loads.

When we compare Figure 5.5 with Figure 5.6, we find little difference between these two protocols in throughput except when the transaction size (TZ) or the system load (TZxMP/DZ) is large, in which case the throughputs are extremely low and the statistics are not reliable anyway.

From the observations of this section, we can conclude that both protocols perform poorly when the average transaction size is large or when the system load is very heavy. In addition, there is no significant difference in performance between the basic timestamp and the multiple version timestamp protocols. More versions of data do not improve significantly the throughput of read-only transactions. When the load is light, the probability of conflict for read-only transactions is very small, therefore more versions of data do not increase the read-only transaction throughput. When the load is heavy, the system is jammed with long update transactions that never finish, thus locking out read-only transactions; therefore more versions of data do not help either.

One may argue that if we do not allow the system to be saturated with long update transactions, then the multiple-version timestamp protocol should perform better than the basic timestamp protocol. We will test this argument in the next section.

5.4 Results of a Modified Model

In the last section, we concluded that there is no significant difference between basic timestamp and multiple-version timestamp protocols in performance, including the throughput of read-only transactions. One may argue that this conclusion is not valid because the simulation model should not have allowed update transactions to jam the system, thus locking out read-only transactions.

To test this argument, we impose the R/W ratio limitation inside the system, instead of in the incoming transaction stream: that is, the ratio of the number of running read-only transactions to the number of running update transactions is always fixed at R/W. All other parameters of the model remain unchanged. The results are shown in Figures 5.7 and 5.8 for the basic and multiple-version timestamp protocols respectively. We include in the figures data from the previous model for comparison. These data are marked by *.

Comparing the data of the modified model to the data of the previous model, we find that by fixing the R/W ratio inside the system, instead of in the incoming transaction stream, the throughputs of read-only transactions increase tremendously when the average transaction size (TZ) is large. The reason is that when the R/W ratio is fixed inside the system, the system can no longer be saturated with long update transactions that never finish. But when the average transaction size is small, fixing the R/W ratio inside the system does not increase significantly the throughputs of read-only transactions. The reason is that the system is never saturated with long update transactions in the first place.

When we compare Figure 5.7 with Figure 5.8, we find no significant difference between the performance of the basic timestamp protocol and the multiple-version timestamp protocol. This contradicts the earlier argument that if the R/W is fixed inside the system instead of in the incoming transaction stream, the multiple-version timestamp protocol should have higher read-only transaction throughputs than the basic timestamp protocol.

The reason for this surprising result is that both timestamp protocols favor read-only transactions. Whenever there is a conflict between an active read-only transaction and an active update transaction, both protocols abort the update transaction. In both protocols, an active read-only transaction is aborted only if it conflicts with a completed update transaction that has a later timestamp, and this occurs rarely because update transactions take much longer to complete. Since read-only transactions rarely get aborted in the basic timestamp protocol, more versions of data make little difference in read-only transaction throughput.

5.5 Timestamp Vs Locking

In this section we compare the performance of the basic timestamp protocol with the performance of the two phase locking protocol.

The simulation model for the two phase locking (LN[1], LN[2], LN[3]) is similar to the timestamp model except that the two phase locking is substituted for the basic timestamp protocol. We show part of the simulation results, specifically the throughput, in Figure 5.9. The unit of the throughputs is the number of data requests completed per time unit, excluding requests aborted.

Comparing Figure 5.9 with Figure 5.5, we find that when the average transaction size (TZ) is small, the basic timestamp protocol outperforms the two phase locking protocol. But when the average transaction size is relatively large (TZ larger than 16) the two phase locking outperforms the basic timestamp protocol.

To learn why the timestamp protocol outperforms the two phase locking when the average transaction size is small, we examined our previous simulation results on the two phase locking protocol ([Lin2], [NL1], [NL2]). We found that, in the two phase locking protocol, blocked transactions tend to wait for long transactions, even when the average transaction size is small. Sinc long transactions take long periods of time to complete, blocked transactions tend to wait for long periods of time. On the other hand, Figure 5.1 shows that when the average transaction size is small, the probability of the basic timestamp protocol restarting a transaction is very small. Therefore we conclude that when the average transaction size is small, restarting transactions in the basic timestamp protocol is better than blocking transactions in the two phase locking protocol. But the reverse is true when the average transaction size is large, because in the timestamp method thrashing is a serious problem: many transactions are constantly aborted and never finish.

We must caution that this result must be taken in the context of our simulation model assumption. In our model, we do not simulate queueing for CPU, IO devices, and communication lines. Queueing for these devices is captured in a single model parameter, the processing delay, which has an erlangian distribution. To validate the conclusions in a more detailed model, we are currently modeling explicit queueing for these devices. Our preliminary results from the more detailed model seem to reaffirm the conclusions.

5.6 Conclusions

We come to three major conclusions concerning the performance of timestamp concurrency control method.

First, over a wide range of system conditions, the multiple version timestamp method performs only marginally better than the basic timestamp method. When the average transaction size (TZ) is small, read-only transactions complete quickly and rarely conflict with younger update transactions that have completed; therefore more versions of data help only marginally. When the average transaction size is large, the system is jammed with update transactions, and few new read-only transactions can start; thus more versions of data do not improve the throughput of read-only transactions either. When we fixed the R/W ratio inside the system to prevent the system from being saturated with update transactions, the multiple version timestamp protocol still performs only marginally better than the basic timestamp protocol, because read-only transactions complete quickly and rarely conflict with younger transactions that have completed.

The second conclusion is that when the average transaction size (TZ) is small, the basic timestamp protocol outperforms two phase locking protocol. But when the average transaction size is relatively larger, the two phase locking protocol outperforms the basic timestamp protocol.

The third conclusion is that when the average transaction size is small, fixing the ratio of read-only transaction to update transactions inside the system does not improve system performance. But when the average transaction size is relatively large, fixing the R/W ratio inside the system significantly improves the throughput of the system, because this prevents the system from being saturated by long update transactions. This amounts to giving read-only transactions higher priority to enter the system; since read-only transactions complete faster, they also enter faster.

But we caution that these conclusions be taken in the context of the simulation model assumptions. Currently we are altering some of the assumptions to see whether these conclusions remain true, and preliminary results seem to indicate that they are.

5.7 References

Bad[1] Badal, D.Z., et al., "A proposal for Distributed Concurrency Control for Partially Redundant Distributed Database System," 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, 1978.

Ber[1] Bernstein, P., N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Survey, Vol. 13, No. 2, June 1981.

Ell[1] Ellis, C.A., "A Robust Algorithm for Updating Duplicate Databases," 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.

Gal[1] Galler, B.I., Ph.D. Thesis, University of Toronto, 1982

Gar[1] Garcia-Molina, H., "Performance of Update Algorithms For Replicated Data in a Distributed Database," Ph.D. Thesis, Dept. of Computer Science, Stanford University, June 1979.

Lin[1] Lin, W.K., "Performance Evaluation of Two Concurrency Controls Mechanisms in a Distributed Database System," Sigmod-81 International Conference on Management of Data, April 1981, Ann Arbor, MI.

Lin[2] Lin, W.K., et al., "Distributed Database Control & Allocation: Semi-Annual Report," Technical Report, Computer Corporation of America, Cambridge, MA.

Lin[3] Lin, W.K., "Concurrency Control In a Multiple Copy Distributed Database System," 4th Berkeley Workshop on Distributed Data Management and Computer Networks Aug. 1979.

LN[1] Lin, W.K., J. Nolte, "Performance of Two Phase Locking," 6th Berkeley Workshop on Distribute Data Management and Computer Networks, Feb. 16-19, 1982, Pacific Grove, CA.

LN[2] Lin, W.K., J. Nolte, "Read Only Transactions and Two Phase Locking," 2nd Symposium on Reliability in Distributed Software and Database Systems, July 19-21, 1982, Pittsburgh, PA.

LN[3] Lin, W.K., J. Nolte, "Communication Delay and Two Phase Locking," 3rd International Conference on Distributed Computing Systems, Oct. 18-22, 1982, Fort Lauderdale, FL.

Rie[1] Ries, D., "The Effect of Concurrency Control on Database Management System Performance," Ph.D. Thesis, Electronics Research Lab, Univ. of Cal., Berkeley, 1979.

Ros[1] Rosenkrantz, D.J., et al., "System Level Concurrency Control for Distributed Database Systems," ACM Tran on Database System, Vol 3, No. 2, June 1978

Ste[1] Sterns, R.E., D.J. Rosenkrantz, et al., "Distributed Database Concurrency Controls Using Before-Values," Sigmod-81 International Conference on Management of Data, April 1981, Ann Arbor, MI.

Sto[1] Stonebraker, M., et al., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES,' IEEE Tran on Software Engineering, Vol SE-5, No. 3 May 1979.

Tha[1] Thanos, C., et al., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed Database System," Lecture Notes in Computer Science, Ed. G. Goo & J. Hartmanis, Springer-Verlag, NY, 1981.

Tho[1] Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Database," ACM Transaction On Database System, Vol 4, No. 2, June 1979.

read-only transaction
DZ = 4096, R/W = 3/1

MP/TZ	4	16	32
16	0.0016	0.0031	0.0013
32	0.0030	0.0026	0.0017
64	0.0049	0.0027	0.0024

read-only transaction
DZ = 8192, R/W = 3/1

MP/TZ	4	16	32
16	0.0011	0.0015	0.0014
32	0.0015	0.0021	0.0011
64	0.0029	0.0021	

write transaction
DZ = 4096, R/W = 3/1

MP/TZ	4	16	32
16	0.0063	0.0236	0.0244
32	0.0117	0.0329	0.0339
64	0.0239	0.0452	0.0456

write transaction
DZ = 8192, R/W = 3/1

MP/TZ	4	16	32
16	0.0033	0.0165	0.0177
32	0.0067	0.0244	0.0254
64	0.0121	0.0337	

Update Transaction
DZ = 4096, R/W = 0

MP/TZ	4	16	32
16	0.0065	0.0238	0.0248
32	0.0127	0.0333	0.0342
64	0.0227	0.0455	0.0458

Figure 5.1
Average Probability of Restart at Read phase
(Basic TS)
Standard Deviation of Processing Delay = 0.528

Update Transaction DZ = 4096, R/W = 3/1			
MP/TZ	4	16	32
16	0.0063	0.0240	0.0247
32	0.0121	0.0339	0.0343
64	0.0232	0.0453	0.0459

Update Transaction DZ = 8192, R/W = 3/1			
MP/TZ	4	16	32
16	0.0040	0.0165	0.0179
32	0.0077	0.0242	0.0255
64	0.0126	0.0338	0.0346

Update Transaction DZ = 4096, R/W = 0			
MP/TZ	4	16	32
16	0.0065	0.0238	0.0248
32	0.0127	0.0333	0.0342
64	0.0227	0.0455	0.0458

Figure 5.2
Average Probability of Restart at Read Phase
(Multiple version TS)
Standard Deviation of Communications Delay = 0.528

Update Transaction DZ = 4096, R/W = 3/1			
MP/TZ	4	16	32
16	0.033	0.270	0.578
32	0.050	0.459	0.785
64	0.079	0.672	0.886

Update Transaction DZ = 8192, R/W = 3/1			
MP/TZ	4	16	32
16	0.016	0.190	0.4702
32	0.027	0.138	0.6149
64	0.049	0.476	____

Update Transaction DZ = 4096, R/W = 0			
MP/TZ	4	16	32
16	0.031	0.296	0.64
32	0.049	0.462	0.91
64	0.083	0.834	0.94

Figure 5.3
Average Probability of Restart at Write Phase
(Basic TS)
Standard Deviation of Processing Delay = 0.528

Update Transaction
DZ = 4096, R/W = 3/1

MP/TZ	4	16	32
16	0.029	0.267	0.476
32	0.048	0.510	0.695
64	0.080	0.684	0.873

Update Transaction
DZ = 8192, R/W = 3/1

MP/TZ	4	16	32
16	0.014	0.165	0.433
32	0.031	0.310	0.629
64	0.048	0.523	0.768

Update Transaction
DZ = 4096, R/W = 0

MP/TZ	4	16	32
16	0.031	0.296	0.64
32	0.049	0.462	0.91
64	0.083	0.834	0.94

Figure 5.4
Average Probability of Restart at Write Phase
(Multiple version TS)
Standard Deviation of Communications Delay = 0.528

Read-Only Transaction
DZ = 4096, R/W = 3/1

MP/TZ	4	8	16	32
16	11.60	7.30	3.67	0.90
32	23.21	11.80	3.29	0.43
64	42.82	14.90	1.61	0.15

Read-Only Transaction
DZ = 8192, R/W = 3/1

MP/TZ	4	8	16	32
16	11.84	8.6	5.89	1.65
32	23.04	14.4	6.17	1.06
64	45.50	24.5	5.12	____

Update Transaction
DZ = 4096, R/W = 3/1

MP/TZ	4	8	16	32
16	3.72	2.34	1.25	0.23
32	7.57	3.85	0.93	0.10
64	13.80	4.80	0.42	0.03

Update Transaction
DZ = 8192, R/W = 3/1

MP/TZ	4	8	16	32
16	3.96	2.88	1.98	0.44
32	7.64	4.78	1.96	0.29
64	15.29	8.23	1.50	____

Update Transaction
DZ = 4096, R/W = 0

MP/TZ	4	16	32
16	14.15	1.14	0.13
32	26.24	0.97	0.03
64	44.19	0.09	0.01

Figure 5.5
Through-put in Requests per Time Unit
(Basic TS)
Standard Deviation of Processing Delay = 0.528

Read-Only Transaction
DZ = 8192, R/W = 3/1

MP/TZ	4	16	32
16	11.8	3.6	1.3
32	23.2	1.8	0.4
64	43.5	1.6	0.3

Read-Only Transaction
DZ = 89192, R/W = 3/1

MP/TZ	4	16	32
16	11.6	5.7	1.6
32	23.2	6.6	1.2
64	46.0	4.5	1.0

Update Transaction
DZ = 8192, R/W = 3/1

MP/TZ	4	16	32
16	4.0	1.14	0.33
32	7.8	0.56	0.08
64	14.4	0.39	0.07

Update Transaction
DZ = 89192, R/W = 3/1

MP/TZ	4	16	32
16	4.0	1.93	0.53
32	7.5	2.19	0.31
64	15.2	1.28	0.21

Update Transaction
DZ = 4096, R/W = 0

MP/TZ	4	16	32
16	14.15	1.14	0.13
32	26.24	0.97	0.03
64	44.19	0.09	0.01

Figure 5.6
Through-put in Requests per Time Unit
(Multiple Versions TS)
Standard Deviation of Communications Delay = 0.528

R/W	DZ	TZ	MP	Read-Only Thru-Put	Update Thru-Put	Probability of Restart (Read-Only)	Probability of Restart(Update During Read-Phase)	Probability of Restart(Update During Write-Phase)
3/1	8192	4	16	10.7	4.8	.0007	.0040	.0164
* same as above				11.8	4.0	.0011	.0033	.0160
3/1	8192	4	64	47.0	13.3	.0021	.0147	.0485
* same as above				45.5	15.3	.0029	.0123	.0490
3/1	8192	16	16	10.3	1.17	.0008	.0379	.1907
* same as above				5.89	1.98	.0015	.0165	.1900
3/1	8192	32	32	22.9	.044	.0001	.0250	.8140
* same as above				1.06	.290	.0011	.0254	.6149

* results of the model with W/R ratio fixed in the input stream

Figure 5.7
Basic Timestamp Model
with R/W Fixed Inside the System

R/W	DZ	TZ	MP	Read-Only Thru-Put	Update Thru-Put	Probability of Restart (Read-Only)	Probability of Restart(Update During Read-Phase)	Probability of Restart(Update During Write-Phase)
3/1	8192	4	16	10.9	4.59	0	.0039	.0150
* same as above				11.6	4.00	0	.0040	.0140
3/1	8192	4	64	46.6	13.4	0	.0134	.0494
* same as above				46.0	15.2	0	.0126	.0480
3/1	8192	16	16	10.9	1.62	0	.0147	.1890
* same as above				5.7	1.93	0	.0126	.1650
3/1	8192	32	32	22.9	1.69	0	.0240	.5370
* same as above				1.2	0.31	0	.0255	.6290

* results of the model with R/W fixed in the input stream

Figure 5.8
Multiple Version Timestamp
With R/W Fixed Within the System

Read-Only Through-Put
DZ=4096, R/W=3/1

MP/TZ	4	16	32
16	8.90	5.01	3.04
32	16.18	5.18	2.04
64	26.50	3.62	1.35

Read-Only Through-Put
DZ=8192, R/W=3/1

MP/TZ	4	16	32
16	9.46	8.04	4.29
32	17.67	9.26	3.56
64	33.18	6.64	2.45

Update Through-Put
DZ=4096, R/W=3/1

MP/TZ	4	16	32
16	2.89	1.73	0.92
32	5.34	1.71	0.63
64	8.82	1.23	0.48

Update Through-Put
DZ=8192, R/W=3/1

MP/TZ	4	16	32
16	3.04	2.65	1.43
32	5.80	3.05	1.19
64	11.02	2.13	0.87

Update Through-Put
DZ=8192, R/W=0

MP/TZ	4	16	32
16	11.79	7.07	3.61
32	21.51	6.69	2.98
64	36.30	5.14	2.04

Figure 5.9
Through-Put of Two Phase Locking

SECTION VI

Performance of Distributed Concurrency Control*

Wente K. Lin
Jerry Nolte

* A version of this paper appeared in the Distributed Database System Designers Handbook, prepared for the DDB Control and Allocation Project.

6. Performance of Distributed Concurrency Control

6.1 Introduction

Many factors effect the performance of a distributed concurrency algorithm:

1. IO delay,
2. communication delay,
3. ratio of read-only to write transactions,
4. database size, transaction size,
5. system multiprogramming level,
6. distribution and replication of the database,
7. overhead of deadlock detection,
8. and system load, defined as the product of transaction size and multiprogramming level divided by the database size.

Our simulation study of the performance of distributed concurrency control algorithms shows that four of these factors have more significant impact than the others: IO delay, communication delay, transaction size, and system load. Hence we divide our simulation results into groups and discuss them separately by classifying the system environment as either IO-bound or communication bound, and as either short transaction loaded or long transaction loaded. We consider a system to be IO bound if queueing for IO or CPU resources is a more significant problem than queuing for communication channel; and we consider a system to be communication bound if queuing for communication channel is a more significant problem than queuing for IO and CPU resources. We consider a system to be short transaction loaded if the average number of data items requested by the transactions (or transaction size) is less than 0.05% of the database. The system is long transaction loaded if the average is larger than 0.2% of the database. If the average is between 0.05% and 0.2% of the database, the classification of the system as short transaction loaded or long transaction loaded depends on the system load. Details of the classification can be found in Figure 6.1.

Thus we present four categories of system environments: short transaction loaded and IO bound (SIO), short transaction loaded and communication bound (SCM), long transaction loaded and IO bound (LIO), and long

System Load / Trans Size	< 10%	> 10%
< 0.05%	Short	Short
0.05%<0.2%	Short	Long
> 0.2%	Long	Long

Trans Size: Average number of data items requested by a transaction as a percentage of the database size.

System Load: Trans Size multiplied by the multiprogramming level.

Database Size: Total number of data items in the database.

Figure 6.1 System Classification (Short Loaded or Long Loaded)

transaction loaded and communication bound (LCM). For each of these four environments, we compare the performance of various concurrency control algorithms, taking into consideration the factors that are not used to classify the system environment -- i.e. multiprogramming level, ratio of read-only to write transactions, distribution and replication of the database.

We first describe, in Section 6.2, the distributed DBMS model that we use to evaluate these algorithms. We then define and describe, in Section 6.3, the concurrency control algorithms that we evaluate. We compare these algorithms in Section 6.4.1 through 6.4.4 for each of the four environments. In Section 6.5 we summarize the results of Section 6. Details of the simulation results can be found in the Appendix.

To use this section as a design guide, a system designer must first classify his system environment, using the following three parameters. First, he must decide whether his system environment is IO bound or communication bound. Second, he must estimate the average number of data items, as a percentage of the total number of data items in the database, requested by a transaction (transaction size). Third, he must estimate the average system load, which is the product of the transaction size and the multiprogramming level of the system (number of transactions running concurrently). Using these three parameters and Figure 6.1, the designer can find his system classification. For each classification, he can find the comparison of various distributed concurrency control algorithms in Section 6.4.1 through Section 6.4.4.

6.2 Performance Model

We assume that there are two kinds of transactions: read-only transactions and write transactions (update transactions). Write transactions always read what they write, and write what they read. This assumption may seem restrictive, but it is a good approximation of real applications. Our earlier simulation results [LIN81a] showed that the total number of requests and the ratio of read-only requests to write requests active at any moment in the system have much greater impact on the system performance than the ratio of read-only to write transactions. Moreover our analysis shows that a more general assumption of transactions would not favor any concurrency control algorithm; thus for performance comparison of the algorithms, this assumption would not distort the results. To use the results of this section to evaluate the performance of a system that has transactions reading more than writing, the ratio of read-only to write transactions in the system can be adjusted upward.

A read-only *transaction consists of a sequence of* read-only requests, and each request reads a data item. A write transaction consists of a sequence of write requests (update requests), followed by a two-phase commit. Requests from a transaction are processed sequentially; another request is initiated only after the previous one has been successfully processed.

As previously described, a distributed DBMS consists of TMs, schedulers, and DMs. Each transaction is managed by a TM, which sequences its requests and sends them to the appropriate scheduler to be processed. If the scheduler site is different from the TM site, a <u>communication delay</u> is incurred.

If a request is read-only, the scheduler requests a read lock for the requested data item (assuming that a two phase locking algorithm is used). Depending on the particular concurrency control algorithm used, some lock managers may grant the lock without checking whether the request conflicts with another transaction. Other lock managers may check for the conflict. If a conflict is found, the read-only request waits and incurs a <u>blocking delay</u>. Depending on the concurrency control algorithm used, the scheduler may initiate a deadlock detection when blocking occurs, thus incurring processing and possibly communication overhead. When the lock for the requested data item is obtained, the scheduler sends the read-only request to the appropriate DM, and the read-only request incurs a processing delay. A read-only transaction ends after all its requests have been successfully processed.

A write request is processed in a manner similar to a read request, except that successful processing of all write requests of a transaction is always followed by a two-phase commit, and a write transaction ends after the two-phase commit is successfully processed (two-phase commit is the only reliability algorithm that we use in our simulation of concurrency control algorithm).

If timestamp based algorithms are used, a timestamp is assigned to each transaction, and requests from the transaction inherit the transaction timestamp. Each data item also has read and write timestamps that record the timestamps of the transactions that last read from (or write into) the data item. For all the timestamp algorithms that we have evaluated, the scheduler always resides at the site of a DM, and a request is always sent to the scheduler at the site where the data is to be accessed. When a scheduler receives a request, it compares the timestamp of the request with the read and write timestamp(s) of the data item, and it may or may not delay the request, depending on the particular algorithm used. If the request is not blocked, it is sent to the DM at the scheduler site, and the request incurs a processing delay.

We simulate both IO bound and communication bound system environments. In the IO bound environment, we explicitly simulate queuing for local processing, which combines cpu and IO processing. We differentiate between local processing of simple messages, such as lock request, lock release, and deadlock detection, and local processing of data requests. The latter needs more processing time than the former. In the IO bound environment, we do not simulate queuing for communication channels. Communication delay is simply simulated by a delay drawn from a probabilistic distribution.

In the communication bound environment, we explicitly simulate queuing for communication channels, but not for local processing resources. In some cases, we differentiate between message and data transmission. The latter takes longer than the former. We simulate local delay (combining IO and cpu processing) by drawing a random number from a probabilistic distribution.

The performance parameters that we use to compare distributed concurrency control algorithms include _read throughput_, _write throughput_, _average read response time_, and _average write response time_. Read throughput is the number of read-only requests successfully completed per time unit; read-only requests processed and subsequently aborted are not included. The write throughput is similarly defined. Read response time is measured from the time a read-only request is initiated by a TM to the time when the next read-only request of the same transaction is initiated by the same TM. Thus, it may include communication delay, blocking delay, and processing delay. Average read response time averages over the response times of all successfully completed read-only requests. Average write response time is similarly computed.

In addition to blocking delay, communication delay, and processing delay, other factors also affect average response times and throughputs (e.g., transaction abortion, deadlock detection, and multiple versions of data). The concurrency control algorithms evaluated in this section can be differentiated by the way they trade off these factors. Some algorithms trade longer blocking delay for fewer transaction abortions, and others trade reversely. Some trade more communication delay for less blocking delay, and others trade reversely. We describe these algorithms in the next section. In Section 6.4, based on the total throughput, we compare and rank these algorithms. Detailed data of the performance parameters can be found in the Appendix.

6.3 Description of Algorithms

The algorithms that we will consider are listed below. Selection of these algorithms is based on our earlier heuristic evaluation reported in [BERN81a]. The selected algorithms were shown to perform better than the algorithms discarded. Names of some algorithms are linked by the conjunctive "and" (e.g. Primary Site and Primary Site). The term before the conjunctive describes the method used for read requests, and the term after the conjunctive describes e method used for write requests. These algorithms are described briefly in this section and summarized in Figure 6.2. Details of these algorithms can be found in the references.

1. Primary Site and Primary Site Two Phase Locking (C-C)
2. Primary Copy and Primary Copy Two Phase Locking (P-P)
3. Basic and Basic Two Phase Locking (B-B)
4. Basic and Primary Copy Two Phase Locking (B-P)
5. Basic and Primary Site Two Phase Locking (B-C)
6. DDM Multiple Version and Optimistic Two Phase Locking (DDM)
7. Basic and Optimistic Two Phase Locking (Opm)
8. Majority Consensus Timestamp (Maj)
9. Wait-Die Two Phase Locking (Die)
10. Basic Timestamp (BaT)
11. Multiple Version Timestamp (MvT)
12. Dynamic Timestamp (Dyn)

The SDD-1 algorithm is not explicitly covered because the Dynamic Timestamp algorithm is an improved version of it ([LIN79, [LIN81]). Neither is the Conservative Timestamp algorithm covered, because this algorithm essentially executes transactions serially in timestamp order. Thus it can perform better than other algorithms only when the transaction size is very large and the system load is extremely heavy and concurrent execution of transactions becomes counterproductive.

The <u>Primary Site and Primary Site</u> method is essentially a centralized two-phase locking method. All requests for read locks and write locks are sent to and processed by a designated primary site, which may use backup sites to improve resiliency. This method trades fewer transaction abortions for more transaction blocking, and it checks for lock conflict as early as possible. It detects deadlock as early as possible, and it avoids distributed deadlock detection; but it has a bottleneck at the primary site.

The <u>Primary Copy and Primary Copy</u> method is a generalized version of the Primary Site and Primary Site method. All requests for read locks and write locks are sent to and processed by a designated primary copy site. However, primary copy sites for different data items may be different, thus distributed deadlock may occur. This method also trades fewer transaction abortions for more transaction blocking, and it checks lock conflict as early as possible. It requires distributed deadlock detection, but it may delay deadlock detection to reduce communication overhead.

The Basic and Basic method sets read locks and reads data locally if a local copy is available; otherwise it locks and reads the closest copy. It sets write locks globally. For each update request, an update lock is requested from all copies, and the update request is granted only after locks from all copies are obtained. This method trades faster read-only transaction response time for slower write transaction response time. It also trades more transaction blocking for fewer transaction abortions. It checks for lock conflict and deadlock as early as possible, and at the expense of more communication overhead.

The Basic and Primary Copy method processes read requests as the previous method does, but it requests write locks only from a designated primary copy. This method checks for most lock conflict as soon as possible, but it may delay distributed deadlock detection to reduce communication overhead. This method also trades fewer transaction abortions for more transaction blocking.

The Basic and Primary Site method is similar to the last method except that update lock requests are sent to a central site instead of to several primary copy sites. Thus deadlock detection is more centralized than in the previous method, and overhead is more centralized at the primary site.

The DDM [CHAN82a, CHAN82b] method avoids conflict between read requests and update requests by keeping several versions of each data item. For each update request, DDM locks locally (if a local copy exists, or locks the closest copy). The update lock is propagated to other copies at transaction end. Detection of most conflicts among update requests is delayed until transaction end. Thus blocking delay is minimized for most write transactions at the expense of more transaction abortions at transaction end.

The Basic and Optimistic method sets read and update locks locally, if a local copy exists; otherwise it locks the closest copy. The update lock is propagated to all copies when the transaction that holds the update lock ends. Thus, distributed lock conflict checking and deadlock detection is delayed until a transaction ends. This algorithm reduces transaction blocking delay at the expense of more transaction abortions.

The Majority Consensus algorithm is similar to the Basic Optimistic algorithm. Each transaction has two phases: a read phase and a commit phase. During the read phase, a transaction reads locally if a local copy exists; otherwise it reads the closest copy. Timestamps of data items read by a transactions are recorded. During the commit phase, both read-only and update transactions must be certified by comparing the timestamps of the data read by each transaction to the transaction timestamp. Because of the certification step, read-only transactions require more communication overhead in this algorithm than in the Basic Optimistic algorithm. The details of the algorithm can be found in [BERN81a,THOM79]. If the algorithm is modified to favor read-only transactions so that read-only transactions need no certification, then it requires no more communication overhead than the Basic Optimistic algorithm. This algorithm checks for lock conflicts as late as possible, and it trades less transaction blocking for more transaction abortions.

In the Wait-Die algorithm, a unique sequence number is attached to every transaction. A transaction always locks locally if a local copy is available; otherwise it locks the closest copy. The locks are propagated to other copies when the transaction commits. Whenever a transaction is blocked by another transaction, the algorithm compares the sequence numbers of the two transactions. If the blocked transaction has a lower priority sequence number, it waits, otherwise it aborts. This algorithm checks local lock conflict as soon as possible, but it checks distributed conflict at transaction end. It has no transaction deadlock (at the expense of more transaction abortions).

In the Basic Timestamp method, a read and a write timestamp are attached to each data item of the database. Each transaction that reads or updates the data item updates its read or write timestamp. Conflict is detected by comparing the timestamp of the transaction that reads or writes a data item with the timestamps of the data item, and not by comparing the timestamps of two transactions as done by the Wait-Die algorithm. This algorithm is similar to the Wait-Die algorithm because it also avoids transaction deadlock. Unlike the Wait-Die algorithm, it has no blocking delay and possibly has more transaction abortions. This algorithm may have fewer transaction abortion than the Wait-Die algorithm when most transactions are read-only, because it allows two transactions (a read-only and a write) to access the same data item simultaneously.

The Multiple Version Timestamp algorithm is a generalization of the previous algorithm. It keeps several versions of each data item in order to reduce conflict between read-only transactions and update transactions. Thus, this method trades more overhead of maintaining multiple data versions for fewer transaction abortions.

The Dynamic Timestamp algorithm [LIN79, LIN81] is an improved version of SDD-1 algorithm; it is unique among all the algorithms that we will compare for the following reasons. It requires *transaction* timestamps but not data item timestamps. It does not avoid transaction blocking, thus it trades more transaction blocking for fewer transaction abortions. But it uses preanalysis of transactions to reduce unnecessary transaction blocking. This algorithm may require a lot of communication overhead when many null write messages are needed [BERN82, LIN79, LIN81], and its performance may depend on system load [LIN81]. Thus it may perform poorly in some system environments.

The principal characteristics of these algorithms are summarized in Figure 6.2.

	B-B	P-P	C-C	B-P	BaT	MvT	DDM	Opm	Maj	Dis	Dyn
blocking/abortion	b	b	b	b	a	a	a	a	a	m	b
lock conflict check	s	s	s	s	s	s	x	x	l	x	s
deadlock detection	s	l	s	l			l	l	l		
Scheduler	2	2	2	2	t	t	2,c	2,c	c	2	t
Location of Scheduler	d	d	cn	d	d	d	d	d	d	d	d
Data Replication	n	p	p	p	p	p	p	p	v	p	n

b: transaction blocking is preferred.
a: transaction abortion is preferred.
m: both blocking and abortion are used.
s: conflict or deadlock is checked as soon as possible.
l: conflict or deadlock is checked as late as possible.
x: local conflict is checked as soon as possible, but
  distributed conflict is checked at transaction end.
" ": the item does not apply.
2: two-phase locking scheduler.
t: timestamp scheduler.
c: certifier scheduler.
2,c: mixed 2-phase locking and certifier scheduler.
cn: centralized.
d: distributed.
n: do nothing.
p: primary copy.
v: voting.

Figure 6.2 Summary of Concurrency Control Algorithms

6.4 Performance Evaluation

6.4.1 Short Transaction Loaded & IO Bound

In this section we compare the performance of distributed concurrency control algorithms in a system environment in which most transactions are relatively short and IO resource is the performance bottleneck. The comparison of these algorithms is summarized in Figure 6.3. The comparison is based on actual simulation results except for the Wait-Die, Majority Consensus Timestamp, and Dynamic Timestamp algorithms. The evaluation of the Wait-Die algorithm is based on its similarity to the Basic Timestamp algorithm; the evaluation of the Dynamic Timestamp algorithm is based on the results of [LIN81]; and the evaluation of the Majority Consensus Timestamp algorithm is based on its similarity with the Basic Optimistic algorithm.

Figure 6.3 shows that five algorithms perform better than others: the Ba Timestamp, Multiple Version Timestamp, DDM, Optimistic, and Wait-Die algorithms.

In the short transaction loaded and IO bound environment, we found that transaction abortion is a better strategy than transaction blocking (i.e. it is better to abort than to wait). The abortion strategy is used by the Basic Timestamp and Multiple Version Timestamp algorithms, and to a large degree by the Wait-Die algorithm. We also found that it is better to delay lock conflict detection than to detect lock conflict early. Both the DDM and the Basic Optimistic algorithms use the delay strategy.

Although the DDM algorithm uses locking for write transactions, and the Optimistic algorithm uses locking for both read and write transactions, blocking occurs only among local transactions that access data from the same site. Transactions running at different sites never block each other. Write locks are propagated to other sites at transaction end, then conflicts among transactions running at different sites are detected and always result in transaction abortions. Therefore performance of these two algorithms is closer to those of timestamp algorithms than to those of two-phase locking algorithms. However, notice that the

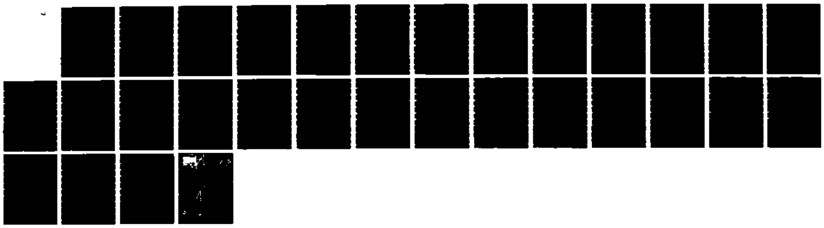
DDM and Basic Optimistic algorithms always abort transactions at transaction end, while the timestamp algorithms may abort transactions at an earlier phase of their execution.

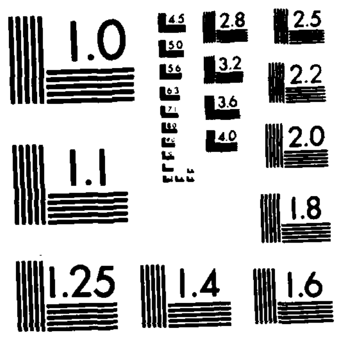
These five algorithms perform equally well in most cases. The timestamp algorithms perform better than the DDM and Basic Optimistic algorithms when the database is fully redundant (thus read-only transactions complete quickly), the R/W ratio is high (probability of conflict among data requests is small), and local delay is large (local blocking delay is large and abortion at transaction end is expensive). However when the database is less redundant, the DDM and Basic Optimistic algorithms perform slightly better than the timestamp algorithms. Both read-only and write transactions require some remote data accesses and take longer to complete, and this causes the probability of conflict among transactions to rise and the timestamp algorithms to abort more transactions.

The Basic Timestamp algorithm performs as well as the Multiple Version Timestamp algorithm, and the latter requires more overhead and storage space for keeping multiple versions of data [LINN83]. Therefore the Basic Timestamp algorithm is preferable to the Multiple Version Timestamp algorithm, unless the multiple versions of data are required in any case for database recovery and resiliency. Similarly, the difference in performance between the DDM and Basic Optimistic algorithms is very small, and the former needs higher overhead and more storage space for keeping multiple versions of data. The Basic Optimistic algorithm is preferable, unless the versions of data are required in any case for database recovery and resiliency.

The Wait-Die algorithm performs slightly worse than the Basic Timestamp algorithm when most transactions are read-only. When a read-only transaction conflicts with a write transaction, the timestamp algorithms never abort the read-only transaction, and they abort the write transaction only when a nonserializable execution may occur. However when most transactions are write transactions, the Wait-Die algorithm is preferred because it performs as well as the Basic Timestamp method and it needs no data item timestamps, which require storage space and processing overhead.

AD-A138 892 DISTRIBUTED DATABASE CONTROL AND ALLOCATION VOLUME 2 2/2
PERFORMANCE ANALYSIS..(U) COMPUTER CORP OF AMERICA
CAMBRIDGE MA W K LIN ET AL. OCT 83
UNCLASSIFIED RADC-TR-83-226-VOL-2 F30602-81-C-0028 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

The Dynamic Timestamp algorithm performs best when most transactions are read-only, communication is fast, database is almost fully redundant, and preanalysis can be done on most transactions. In this environment, the fast protocols, R1, R1a, R1ab, and R1b [LIN79], LIN82] apply to most transactions. Assuming system conditions remain the same except that the database is not redundant, the Dynamic Timestamp algorithm still performs relatively well, because more efficient protocols (R2, R2a, R2ab, and R2b) apply to most transactions. These protocols are not as efficient as the group of R1 protocols, but they are relatively fast compared with R3 protocol. In all other cases, either when the communication is slow or when most transactions update the database, the Dynamic Timestamp algorithm is not efficient.

The Majority Consensus algorithm performs reasonably well, but not as well as the Basic Optimistic algorithm. The Majority Consensus algorithm as proposed in [THOM79] requires extra communication overhead for read-only transactions. If the algorithm is modified to favor read-only transactions, so that read-only transactions need not be certified, then it would perform as well as the Basic Optimistic algorithm.

To summarize, in this environment transaction abortion is a better strategy than transaction blocking, and delayed lock conflict checking is a better strategy than early lock conflict checking.

6.4.2 Short Transactions & Communication Bound

In this section we compare the performance of distributed concurrency control algorithms in a system environment in which most transactions are relatively short and communication channel is the performance bottleneck. The comparison of the algorithms is summarized in Figure 6.4. The comparison is based on actual simulation results except for the Wait-Die, Majority Consensus, and the Dynamic Timestamp algorithms. The evaluation of the Wait-Die algorithm is based on its similarity to the Basic Timestamp algorithm; the evaluation of the Dynamic Timestamp algorithm is based on the results of [LIN81]; and the evaluation of the Majority Consensus algorithm is based on its similarity to the Basic Optimistic algorithm.

			B-B	P-P	C-C	B-P	BaT	MvT	DDM	Opm	Maj	Die	Dyn
R/W	L/C	Red											
low	*	full	6	4	5	3	1	1	1	1	2	1	3
low	low	full	6	4	5	3	1	1	1	1	2	1	3
high	low	full	6	4	5	3	1	1	1	1	3	1	3
high	high	full	4	4	5	3	1	1	2	2	3	2	1
high	high	part	6	6	7	5	3	3	1	2	3	4	2
high	low	part	5	5	6	4	2	2	1	1	2	3	2
low	*	part	6	4	5	4	2	2	1	1	2	2	3

Rank 1 is best and Rank 6 is worst.
Rank numbers have no absolute meaning. They only show relative performance across a row, not across a column.
R/W: Ratio of Read-only transactions to Write transactions
L/C: Ratio of Local delay to Communication delay, excluding queuing delay
Red: Redundancy of the database
* : Does not matter

Figure 6.3 Performance Comparison: Short Transaction Loaded & IO Bound

Figure 6.4 shows that seven algorithms perform better than the others: Basic-Primary Copy, Basic Timestamp, Multiple Version Timestamp, DDM, Basic Optimistic, Wait-Die, and Dynamic Timestamp.

We found that transaction abortion, similar to the SIO environment, is a better strategy than transaction blocking, and that delayed lock conflict detection is a better strategy than early detection. However, because of the communication channel bottleneck, performance of the algorithms that require extra communication messages degrade in some cases.

The Basic Timestamp and Multiple Version Timestamp algorithms perform best in all cases. However, when the database is fully redundant, the DDM and Basic Optimistic algorithms perform just as well. Read-only transactions never incur communication delays, and write transactions incur communication delays only during the commit phase. Therefore transactions finish fast, blocking delay is shorter, and abortion at transaction end is less expensive.

The Majority Consensus algorithm, as proposed in [THOM79], does not perform well because of the extra communication messages required for read-only transactions. If the algorithm is modified to favor read-only transactions, so that read-only transactions need not be certified, the algorithm would perform as well as the Basic Optimistic algorithm.

The Wait-Die algorithm performs just as well as the timestamp algorithms in most cases. However, when most transactions are read-only, the Wait-Die algorithm unnecessarily aborts more read-only transactions than the timestamp algorithms, thus performing worse than the timestamp algorithms.

The DDM algorithm performs as well as the timestamp algorithms when the database is fully redundant. However, when the database is less redundant and most transactions are read-only, its performance degrades as shown in Figure 6.4. When the database is not fully redundant, read-only transactions require one extra communication message, which causes a long delay in a communication bound environment.

The Basic-Primary Copy algorithm performs 10% to 20% worse than the best algorithms in all cases, because it incurs extra communication messages when obtaining locks from the primary copies, and it uses transaction blocking instead of transaction abortion. The Dynamic Timestamp algorithm performs best when most transaction are read-only and can be preanalyzed. In this environment, the most efficient protocols can be used and communication overhead for null-write messages is minimized.

Since the Basic Timestamp algorithm performs as well as the Multiple Version Timestamp algorithm, the former is preferable unless the multiple versions of data are required in any case for database recovery and resiliency. Similar observations apply to the DDM and Basic Optimistic algorithms [LINN83].

Our conclusion is that in this environment abortion is better than blocking, and that delayed lock conflict checking is better than early lock conflict checking. However, some algorithms that use these two strategies may not perform well in some cases because they require extra communication messages.

			B-B	P-P	C-C	B-P	BaT	MvT	DDM	Opm	Maj	Die	Dyn
R/W	L/C	Red											
low	*	full	5	4	4	3	1	1	1	1	3	1	3
high	low	full	5	6	4	3	1	1	1	1	5	2	2
high	high	full	5	6	4	2	1	1	1	1	5	2	2
high	low	part	5	6	7	3	1	1	4	2	5	2	2
low	low	part	5	4	6	2	1	1	2	1	3	1	4
high	high	part	4	5	6	2	1	1	3	1	5	2	2
low	high	part	5	4	6	3	1	1	2	1	3	1	4

Rank 1 is best and Rank 6 is worst.
Rank numbers have no absolute meaning. They only show relative performance across a row, not a column.
R/W: Ratio of Read-only transactions to Write transactions
L/C: Ratio of Local delay to Communication delay, excluding queuing delay
Red: Redundancy of the database
* : Does not matter

Figure 6.4 Performance Comparison: Short Transaction Loaded & Communication Bound

6.4.3 Long Transaction Loaded & IO Bound

In this section we compare the performance of distributed concurrency control algorithms in a system environment in which most transactions are relatively long and IO resource is the bottleneck. The comparison is summarized in Figure 6.5. The comparison is based on actual simulation results except for the Wait-Die and Majority Consensus algorithms. The evaluation of the Wait-Die algorithm is based on its similarity to the Basic Timestamp algorithm; and the evaluation of the Majority Consensus algorithm is based on its similarity to the Basic Optimistic algorithm.

Figure 6.5 shows that three algorithms perform better than the others: Basic Primary, DDM, and Basic-Optimistic.

In this environment (long transactions, heavy system load) transactions conflict with each other more often, but only a fraction of the conflicts lead to transaction deadlocks. Thus, transaction blocking is better than indiscriminate transaction abortion. Moreover, prompt lock conflict-detection is better than procrastination. Lock conflicts that are detected at transaction end always lead to deadlocks. The Basic Primary, DDM, and Basic Optimistic algorithms use the blocking strategy.

The Basic Primary algorithm uses the early lock conflict detection strategy.

The Basic Primary Copy algorithm performs best in this environment because it does not abort a transaction unless it deadlocks, and it detects lock conflicts as soon as they occur. However, when most transactions are read-only, and the database is not fully redundant, the Basic Primary Copy does not perform as well as the DDM and Basic-Optimistic algorithms, because the extra communication messages required by the Basic Primary Copy algorithm for write-locks and deadlock detections does not outweigh the extra transaction abortions by the DDM and Basic-Optimistic algorithm.

The DDM and the Basic Optimistic algorithms perform well in partially redundant databases, because more lock conflicts are detected during the reading phase of transactions and less transactions abort at the commit phase. However, when the database is fully redundant, most lock conflicts are detected during the commit phase, which always leads to deadlocks and transaction abortions, thus resulting in the poorer performance of these two algorithms in this conditions.

The timestamp algorithms do not perform as well as the Basic-Primary method because transaction blocking is better than transaction abortion. However, the timestamp algorithms perform better than the DDM and Basic-Optimistic algorithms, when the database is fully redundant. Read-only transactions incur no communication delay and complete quickly; the read-phase of write transactions also completes quickly. Thus conflict between read-only transactions and write transactions that result in the abortion of write transactions is reduced. In addition, when the database is fully redundant, the timestamp algorithms detect more conflicts at the read-phase, thus aborting more transactions at earlier stages of processing, while the DDM and Basic-Optimistic algorithms detect more conflicts at the commit phase, thus aborting more transactions at their ends. However, when the database is not fully redundant, the DDM and Basic-Optimistic algorithms detect more conflicts at the read-phase, and they abort more transactions at the early stages of processing, thus performing better than the timestamp algorithms.

The Wait-Die algorithm performs as well as the Basic Timestamp algorithm, except when most transactions are read-only. Then the Basic Timestamp algorithm has higher throughput of read-only transactions than the Wait-Die algorithm.

The Majority Consensus algorithm also performs poorly because it delays lock conflict detection until transaction end, thus resulting in many late transaction abortions. In fact, all certifier algorithms that certify transactions at transaction end perform badly in the long transaction environment. The Primary Site & Primary Site (C-C) and the Primary Copy & Primary Copy (P-P) algorithms also perform relatively well when the database is fully redundant. These two algorithms abort fewer transactions than the Basic Timestamp, Multiple Version Timestamp, DDM, and Basic Optimistic algorithms, and the savings in transaction abortions more than make up for the extra communication messages required by the two algorithms. The Basic-Basic algorithm does not perform as well because it requires many more communication messages than other algorithms.

To summarize, in this environment transaction blocking is better than transaction abortion, and early lock conflict detection is better than late detection.

			B-B	P-P	C-C	B-P	BaT	MvT	DDM	Opm	Maj	Die
R/(R+W)	Loc/Com	Redundant										
low	low	full	5	2	2	1	2	2	3	3	4	2
high	low	full	5	2	2	1	2	2	3	3	4	3
low	high	full	5	2	2	1	2	2	3	3	4	2
high	high	full	5	2	2	1	2	2	3	2	4	3
low	low	part	5	2	2	1	3	3	1	1	4	2
high	low	part	5	3	3	2	3	3	1	1	4	3
low	high	part	5	2	2	1	3	3	1	1	4	2
high	high	part	5	3	3	2	3	3	1	1	4	3

Rank 1 is best and Rank 6 is worst.
Rank numbers have no absolute meaning. They only show relative performance across a row, not a column.
R/W: Ratio of Read-only transactions to Write transactions
L/C: Ratio of Local delay to Communication delay, excluding queuing delay
Red: Redundancy of the database
* : Does not matter

Figure 6.5 Performance Comparison: Long Transaction Loaded & IO Bound

6.4.4 Long Transactions & Communication Bound

In this section, we compare the performance of distributed concurrency control algorithms in a system environment in which most transactions are long and communication channel is the bottleneck. The comparison of these algorithms is summarized in Figure 6.6. The comparison is based on actual simulation results except for the Wait-Die and Majority Consensus algorithms. The evaluation of the Wait-Die algorithm is based on its similarity to the Basic Timestamp algorithm; and the evaluation of the Majority Consensus algorithm is based on its similarity to the *Basic Optimistic algorithm*.

Figure 6.6 shows that six algorithms perform better than the others: Basic Timestamp, Multiple Version Timestamp, DDM, Basic Optimistic, Basic Primary, and Wait-Die.

In this system environment (long transactions, heavy system load, and long communication delay) transactions conflict with each other more often, but only a fraction of the conflicts lead to deadlocks; thus, transaction blocking is better than indiscriminate transaction abortion. Moreover, early lock conflict detection is better than procrastination.

Lock conflicts detected at transaction end always lead to deadlocks. The Basic Primary, DDM, Basic Optimistic, and to certain degree the Wait-Die algorithms use the blocking strategy; and the Basic Primary and Wait-Die algorithms detect lock conflicts as early as possible. In addition, because of long communication delay, algorithms requiring extra communication messages may not perform well even if they use transaction blocking instead of transaction abortion. The DDM and the Basic Primary algorithms require extra communication messages in some cases.

The Basic Primary Copy algorithm performs the best when the database is not fully redundant because it requires no more communication messages than the other algorithms, and because it causes fewer unnecessary transaction abortions. Even when the database is not fully redundant, if most transactions are write transactions and local delay is high relative to the communication delay, the Basic Primary Copy algorithm still performs better than the Basic Timestamp, Multiple Version Timestamp, DDM, and Basic-Optimistic algorithms, because the latter abort write transactions frequently. However, when the database is fully redundant, the Basic Primary Copy algorithm requires more communication messages than the Basic Timestamp, Multiple Version Timestamp, DDM, and Basic Optimistic algorithms. Thus, except for the cases above, the extra communication messages required by the Basic Primary Copy algorithm make its performance worse than that of the Basic Timestamp, Multiple Version Timestamp, DDM, and Basic-Optimistic algorithm in this communication bound environment.

The timestamp based algorithms perform best when the database is fully redundant, then read-only transactions incur no communication delay and complete quickly. The read phase of write transactions also completes quickly. When read-only transactions and the read phase of write transactions complete quickly, conflicts between read-only and write transactions that result in abortion of the write transactions is reduced. Thus, unnecessary transaction abortion is reduced.

The DDM method avoids conflicts between read-only transactions and write transactions, but it pays with more abortions of write transactions at transaction end. Thus, when most transactions are read-only, it performs very well. The higher throughput of read-only transactions make up for the extra abortion of write transactions. Notice that DDM requires a extra round of communication messages for read-only transactions when the database is not fully redundant. Then its performance degrades.

The Basic-Optimistic algorithm also performs well when most transactions are read-only; then read-only transactions and the read phase of write transactions complete quickly. Otherwise it performs poorly because the system is eventually saturated with many long write transactions that later abort.

The Wait-Die algorithm performs as well as the Basic Timestamp algorithm when most transactions are write transactions, but not as well when most transactions are read-only transactions. Since the Wait-Die algorithm needs no overhead for maintaining data item timestamps, it is preferable to the timestamp based algorithms if most transactions are write transactions.

The Basic & Basic, Primary Copy & Primary Copy, and Primary Site & Primary Site algorithms perform poorly because they require more communication messages than other algorithms. Communication overhead is expensive in this communication bound environment.

To summarize, in this environment transaction blocking is better than transaction abortion, and early lock conflict detection is better than late detection. However, some algorithms that use these two strategies may not perform well in some cases because they require extra communication messages.

6.5 Conclusion

We found that five of the twelve algorithms perform best in various system environments: Basic Timestamp, Multiple Version Timestamp, DDM, Basic Optimistic, and Basic-Primary algorithms.

When most transactions are short, concurrency control algorithms that abort conflicting transactions (such as Basic Timestamp, Multiple Version Timestamp algorithms) perform better than algorithms that block conflicting transactions (such as the Basic Primary algorithm). In this

R/(R+W)	Loc/Com	Redundant	B-B	P-P	C-C	B-P	BaT	MvT	DDM	Opm	Maj	Die
low	low	full	6	5	5	6	1	1	5	4	6	1
high	low	full	6	5	5	4	1	1	3	3	6	2
low	high	full	6	5	5	1	2	2	4	3	6	2
high	high	full	6	5	5	4	2	2	1	3	6	3
low	low	part	6	5	5	1	3	3	2	3	6	3
high	low	part	6	5	5	1	2	2	2	1	6	3
low	high	part	6	5	5	1	2	2	4	3	6	2
high	high	part	6	5	5	2	3	3	1	2	6	3

Rank 1 is best and Rank 6 is worst.
Rank numbers have no absolute meaning. They only show relative performance across a row, not a column.
R/W: Ratio of Read-only transactions to Write transactions
L/C: Ratio of Local delay to Communication delay, excluding queuing delay
Red: Redundancy of the database
* : Does not matter

Figure 6.6 Performance Comparison: Long Transactions & Communication Bound

environment, transactions conflict rarely; and when they do conflict, the blocking transactions tend to be longer than the average transaction size and blocking delay long [LINN83]. If a two-phase locking algorithm must be used, algorithms that delay lock conflict checking (such as the DDM and the Basic Optimistic algorithms) perform better than those that expedite lock conflict checking (such as the Basic Primary algorithm). Unless the communication bandwidth is very high, communication delay can devastate system performance; thus, the designer should reduce communication delay by locally controlling and accessing data as much as possible.

The issue of balancing communication delay against data distribution and replication is part of the complex problem of distributed database design. Distributed database design must also take into account the issues of distributed query processing and distributed database reliability, and is beyond the scope of this handbook.

Behavior of systems that have long transactions is very different from that of systems that have short transactions. Long transactions degrade system performance very quickly because they have more transaction conflicts. Since only a fraction of these conflicts results in deadlocks, concurrency control algorithms that use transaction blocking often perform better than those that use transaction abortion indiscriminately. Moreover, concurrency algorithms that detect transaction conflict earlier often perform better than those that detect transaction conflict later. The effect of communication delay on the performance of a system that has long transactions is even more devastating than the effect on a system that has short transactions. Thus the designer must reduce communication delay as much as possible by controlling and accessing data locally.

However, no matter which concurrency algorithm the desig[illegible]er uses, a system that has long transactions always performs worse t[illegible] system that has short transactions. The designer should design tran[illegible]ions to access as much data in parallel as possible, and to break long transactions into shorter transactions. Long transactions that cannot be broken into shorter ones must be executed in background mode.

Our performance study shows that no one algorithm performs best in all system and application environments. If the system environment is stable, the database designer can select one algorithm that performs best in the environment. If the system environment is not stable, the database designer can assign different weights to different environments according to how often the system stays in each environment. The database designer then selects the algorithm that has the best weighted average performance.

From the results, we can also conclude that the best algorithm would be one that could be adjusted by the system administrator according to the environment. The administrator would adjust the algorithm to use transaction abortion and delay lock conflict detection whenever transactions are short, and to use transaction blocking and detect lock conflicts as soon as possible whenever transactions are long. The adjustable algorithm would also alternate, depending on the load on the communication channel, between algorithms that have more localized control and algorithms that have more distributed control.

6.6 References

Lin[1] Lin, W.K., "Concurrency Control in a Multiple Copy Distributed Database System," 4th Berkeley Workshop on Distributed Data Management and Computer Networks, Aug. 1979., Berkeley, CA.

Lin[2] Lin, W.K., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed Database System," ACM SIGMOD-81 International Conference on Management of Data, April 1981, Ann Arbor, MI.

LN[1] Lin, W.K., J. Nolte, "Performance of Two Phase Locking," 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Feb. 1982, Pacific Grove, CA.

LN[2] Lin, W.K., J. Nolte, "Read-Only Transaction and Two Phase Locking," 2nd IEEE Symposium on Reliability in Distributed Software and Database Systems, Jul. 1982, Pittsburgh, PA.

LN[3] Lin, W.K., J. Nolte, "Communication Delay and Two Phase Locking," 3rd International Conference on Distributed Computing Systems, Oct. 1982, Fort Lauderdale, FL.

[Tho] Thomas, R.H. "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM Trans. on Database Systems, Vol. 4, No. 2, June 1979, pp. 180-209.

SECTION VII

Conclusion

Wente K. Lin

7. Conclusion

The DDB Control and Allocation Project has set out to achieve the following objectives:

1. Review the distributed concurrency control research published in the literature and incorporate that research into the taxonomy of the distributed database concurrency control algorithms. Based on this taxonomy, we would develop a new framework for distributed database concurrency control.

2. Develop new distributed database concurrency control algorithms using the framework developed in 1.

3. Simulate the performance of the distributed database concurrency control algorithms that are found to be dominant in the previous study.

4. Build an analytical model of distributed database concurrency control.

5. Survey the current studies of reliability and recovery of distributed database systems and the analysis of published algorithms.

6. Develop a framework for reliability and recovery of distributed database systems.

7. Consolidate the results of the previous tasks into a system designer's handbook.

We have achieved these objectives, and the results are described in this final technical report.

The first objective is achieved by means of the framework discussed in Section II of Volume I. The framework facilitates the taxonomy of distributed concurrency control algorithms by identifying the essential component functions of distributed concurrency control mechanisms. This framework is an excellent basis for further research in the standardization of distributed concurrency control architecture.

The second objective of developing new algorithms using this framework is achieved by the new distributed concurrency control algorithms described in Section III of Volume I. In this section, new algorithms that store and use older versions of data items are described.

The third objective of simulating and evaluating the performance of distributed database concurrency control algorithms is achieved by a series of reports in the second volume. Sections II through V report the relationship between the performance of various algorithms and the system parameters. Section VI summarizes the simulation results and compares the performance of twelve algorithms. The results of the second volume serve as an excellent basis for designing a distributed database designer's aid. The Designer Aid would help the system designer to design distributed transactions, partition the database into fragments, replicate and distribute the fragments, and choose the concurrency control algorithm that performs best in his system environment.

The forth objective of analytical modeling of the distributed concurrency control algorithms is achieved through the analytical models described in Sections IV and V of Volume I.

The survey/study of reliability and recovery of distributed database systems that achieves the fifth objective is reported in Sections VI through IX of Volume I and in the third semiannual technical report. Because the subject is relatively unexplored, only a few algorithms were reported. To discover new algorithms further research is needed.

A framework for the reliability and recovery of a distributed database system achieving the sixth objective is described in Section VII of Volume I. This framework captures the essential components of existing reliability and recovery algorithms. But, because research on this subject is in its primitive stage, more research is needed to refine the framework and to use it to develop more efficient algorithms. Moreover, the refined framework should become a basis for standardizing of distributed reliability and recovery architecture.

Finally, these results have been summarized in a separate distributed database system designer's handbook. This handbook can help the designer to select a distributed concurrency control algorithm that performs best in his system environment. Of course, an automated tool would be more helpful to the designer. The automated tool would receive information from the designer about his system (e.g., system and application parameters) and it would output to the designer information about how to best design his system.

Overall we have accomplished what we set out to do; and in the process we came to understand more fully the mechanism of concurrency control, reliability, and recovery of distributed database systems. The next step is to translate these results and this understanding into a practical, integrated set of tools that aid distributed database designers, and into a standard architecture of distributed DBMS that facilitates the interconnection of different DBMSs.

APPENDIX A

Wente K. Lin

Jerry Nolte

A.

Notations used in the appendix are explained here and in the figures.

READ THROUGHPUT: average number of read-only requests successfully processed per unit time (excluding requests processed and subsequently aborted).

WRITE THROUGHPUT: Average number of write requests successfully processed per unit time (excluding requests processed and subsequently aborted).

Average Response Per Read Request: average time required to process a read-only request.

Average Response Per Write Request: average time required to process a write request.

Basic Basic	Basic and Basic algorithm.
Prmry Prmry	Primary Copy and Primary Copy algorithm.
Cntrl	Primary Site and Primary Site algorithm.
Basic Prmry	Basic and Primary Copy algorithm.
Basic Cntrl	Basic and Primary Site algorithm.
Basic Tstmp	Basic Timestamp algorithm.
Mltpl Versn	DDM Multiple Version and Optimistic algorithm.
Basic Optms	Basic and Optimistic algorithm.

TZ=4, DZ=8192

MP	R/(R+W)	IO/Comm	no. S1	of S2	copy S3	Basic Basic	Prmry prmry	Cntrl	Basic Prmry	Basic Cntrl	Basic Tstmp	Mltpl Versn	Basic Optms
*	25%	.2	1	1	1	0.8	1.2	1.1	1.6		3.1	3.1	3.3
*	50%	.2	1	1	1	2.2	2.6	3.1					
*	75%	.2	1	1	1	6.6	4.5	8.3	15.		30	29	30
*	75%	1	1	1	1	6.3	4.4	5.9					
*	75%	.5	1	1	1	6.9	4.6	7.0	15.				
*	50%	.5	1	1	1				5.				
*	25%	.5	1	1	1				1.7				
*	75%	2	1	1	1				8.		9.8	9.8	9.2
*	50%	2	1	1	1				3.9				
*	25%	2	1	1	1				1.5		2.3	2.4	2.5
*	75%	.2	2/3	2/3	2/3	5.0	4.6	3.4	7.8		10.3	6.1	8.7
@	75%	.2	2/3	2/3	2/3			5.1					
#	75%	.2	2/3	2/3	2/3			5.9					
*	75%	.2	1/2	1/2	1/2	4.9	4.7	3.1					
*	50%	.2	2/3	2/3	2/3	2.4	2.8						
*	50%	.2	1/2	1/2	1/2	2.7	2.8						
*	25%	.2	2/3	2/3	2/3	0.9	1.3	.92	1.8		2.0	1.9	2.1
*	25%	.2	1/2	1/2	1/2	1.1	1.3	.88					
*	75%	.5	2/3	2/3	2/3			3.5	7.8				
*	75%	.5	1/2	1/2	1/2			3.0	6.4				
*	75%	1	2/3	2/3	2/3			3.2					
*	75%	1	1/2	1/2	1/2			3.0					
*	50%	.5	2/3	2/3	2/3				3.8				
*	50%	.5	1/2	1/2	1/2				3.6				
*	25%	.5	2/3	2/3	2/3				1.5				
*	25%	.5	1/2	1/2	1/2				1.5				
*	75%	2	2/3	2/3	2/3				6.2		6.9	5.4	6.6
*	75%	2	1/2	1/2	1/2				5.5				
*	50%	2	2/3	2/3	2/3				3.5				
*	50%	2	1/2	1/2	1/2				3.1				
*	25%	2	2/3	2/3	2/3				1.4		1.7	1.6	1.7
*	25%	2	1/2	1/2	1/2				1.4				
*	25%	.2	1	1/2	1/2			1.3					
*	50%	.2	1	1/2	1/2			3.5		5.0			
*	75%	.2	1	1/2	1/2			9.0		16.			
*	25%	.5	1	1/2	1/2			1.2					
*	50%	.5	1	1/2	1/2			3.3					
*	75%	.5	1	1/2	1/2			7.3					
*	25%	1	1	1/2	1/2			1.2					
*	50%	1	1	1/2	1/2			3.1		4.7			
*	75%	1	1	1/2	1/2			6.0		8.9			
*	25%	2	1	1/2	1/2			1.1					
*	50%	2	1	1/2	1/2			2.6					
*	75%	2	1	1/2	1/2			4.5					
@	50%	.2	1	1/2	1/2			4.8		5.4			
@	75%	.2	1	1/2	1/2			14.		17.			
@	50%	1	1	1/2	1/2			4.5		5.2			
@	75%	1	1	1/2	1/2			9.7		12.			
#	50%	.2	1	1/2	1/2			5.5		5.7			
#	75%	.2	1	1/2	1/2			16.		17.			
#	50%	1	1	1/2	1/2			5.4		5.7			
#	75%	1	1	1/2	1/2			13.		14.			

* Multiple programming levels at the three site are 10/11/11.
@ Multiple programming levels at the three site are 16/8/8.
Multiple programming levels at the three site are 24/4/4.
TZ : Average no. of requests per transaction (transaction size).
DZ : Total number of data items in the database (database size).
MP : Multiprogramming level.
R/(R+W) : Percentage of transactions that are raed-only.
IO/Comm : Ratio of local delay to communication delay
 - (excluding queueing delay).
No. of Copy : Fraction of the database residing at sites S1, S2, & S3.

Figure A.1 READ THROUGHPUT: Short Transaction
Loaded & Communication Bound

TZ=4, DZ=8192

MP	R/(R	IO/	no. of	copy		Basic	Prmry	Cntrl	Basic	Basic	Basic	Mltpl	Basic
	+W)	Comm	S1	S2	S3	Basic	Prmry	Total	Prmry	Cntrl	Tstmp	Versn	Optms
*	25%	.2	1	1	1	2.2	3.5	3.4	5.1		9.6	9.4	9.4
*	50%	.2	1	1	1	2.2	2.6	3.0					
*	75%	.2	1	1	1	2.2	1.5	2.5	5		9.5	9.6	9.3
*	75%	1	1	1	1	2.1	1.5	1.9					
*	75%	.5	1	1	1	2.2	1.4	2.2	4.9				
*	50%	.5	1	1	1				4.9				
*	25%	.5	1	1	1				5.0				
*	75%	2	1	1	1				2.7		3.2	3.1	3.1
*	50%	2	1	1	1				4.0				
*	25%	2	1	1	1				4.5		7.4	7.2	7.3
*	75%	.2	2/3	2/3	2/3	1.7	1.6	1.3	2.7		3.4	2.0	3.1
@	75%	.2	2/3	2/3	2/3			1.7					
#	75%	.2	2/3	2/3	2/3			1.9					
*	75%	.2	1/2	1/2	1/2	1.7	1.6	1.0					
*	50%	.2	2/3	2/3	2/3	2.4	2.7						
*	50%	.2	1/2	1/2	1/2	2.6	2.8						
*	25%	.2	2/3	2/3	2/3	2.7	3.6	2.7	4.8		6.2	5.4	6.0
*	25%	.2	1/2	1/2	1/2	3.2	3.8	2.6					
*	75%	.5	2/3	2/3	2/3			1.1	2.6				
*	75%	.5	1/2	1/2	1/2			.98	2.2				
*	75%	1	2/3	2/3	2/3			1.2					
*	75%	1	1/2	1/2	1/2			.93					
*	50%	.5	2/3	2/3	2/3				4.0				
*	50%	.5	1/2	1/2	1/2				3.7				
*	25%	.5	2/3	2/3	2/3				4.7				
*	25%	.5	1/2	1/2	1/2				4.7				
*	75%	2	2/3	2/3	2/3				2.0		2.3	1.7	2.2
*	75%	2	1/2	1/2	1/2				1.8				
*	50%	2	2/3	2/3	2/3				3.3				
*	50%	2	1/2	1/2	1/2				3.2				
*	25%	2	2/3	2/3	2/3				4.3		5.2	4.8	5.2
*	25%	2	1/2	1/2	1/2				4.2				
*	25%	.2	1	1/2	1/2			3.9					
*	50%	.2	1	1/2	1/2			3.5		5.0			
*	75%	.2	1	1/2	1/2			2.9		5.2			
*	25%	.5	1	1/2	1/2			3.8					
*	50%	.5	1	1/2	1/2			3.2					
*	75%	.5	1	1/2	1/2			2.5					
*	25%	1	1	1/2	1/2			3.6					
*	50%	1	1	1/2	1/2			3.0		4.7			
*	75%	1	1	1/2	1/2			2.0		3.0			
*	25%	2	1	1/2	1/2			3.4					
*	50%	2	1	1/2	1/2			2.6					
*	75%	2	1	1/2	1/2			1.5					
@	50%	.2	1	1/2	1/2			4.8		5.4			
@	75%	.2	1	1/2	1/2			4.4		5.4			
@	50%	1	1	1/2	1/2			4.4		5.2			
@	75%	1	1	1/2	1/2			3.2		4.1			
#	50%	.2	1	1/2	1/2			5.5		5.7			
#	75%	.2	1	1/2	1/2			5.4		5.6			
#	50%	1	1	1/2	1/2			5.3		5.6			
#	75%	1	1	1/2	1/2			4.3		5.0			

* Multiple programming levels at the three site are 10/11/11.
@ Multiple programming levels at the three site are 16/8/8.
Multiple programming levels at the three site are 24/4/4.
TZ : Average no. of requests per transaction (transaction size).
DZ : Total number of data items in the database (database size).
MP : Multiprogramming level.
R/(R+W) : Percentage of transactions that are read-only.
IO/Comm : Ratio of local delay to communication delay
(excluding queueing delay).
No. of Copy : Fraction of the database residing at sites S1, S2, & S3.

Figure A.2 WRITE THROUGHPUT: Short Transaction
Loaded & Communication Bound

TZ=4, DZ=8192

MP	R/(R	IO /	no.	of	copy	Basic	Prmry	Cntrl	Basic	Basic	Basic	Mltpl	Basic
	+W)	Comm	S1	S2	S3	Basic	Prmry	Total	Prmry	Cntrl	Tstmp	Versn	Optms
*	25%	.2	1	1	1	.31	5.1	4.0	.26		.20	.20	.22
*	50%	.2	1	1	1	.27	4.9	3.3					
*	75%	.2	1	1	1	.26	4.7	2.1	.26		.20	.20	.22
*	75%	1	1	1	1	1.1	4.6	3.3					
*	75%	.5	1	1	1	.55	4.7	2.7	.56				
*	50%	.5	1	1	1				.55				
*	25%	.5	1	1	1				.55				
*	75%	2	1	1	1				2.1		2	2	2
*	50%	2	1	1	1				2.1				
*	25%	2	1	1	1				2.1		2	2	2
*	75%	.2	2/3	2/3	2/3	2.7	4.6	6.2	2.5		2.2	2.6	2.5
@	75%	.2	2/3	2/3	2/3			4.9					
#	75%	.2	2/3	2/3	2/3			4.2					
*	75%	.2	1/2	1/2	1/2	3.6	4.6	7.2					
*	50%	.2	2/3	2/3	2/3	2.7	4.7	6.5					
*	50%	.2	1/2	1/2	1/2	3.7	4.6	7.4					
*	25%	.2	2/3	2/3	2/3	2.8	4.9		2.5		2.7	2.8	2.8
*	25%	.2	1/2	1/2	1/2	3.8	4.7						
*	75%	.5	2/3	2/3	2/3			6.3	2.6				
*	75%	.5	1/2	1/2	1/2			7.2	3.5				
*	75%	1	2/3	2/3	2/3			6.3					
*	75%	1	1/2	1/2	1/2			7.3					
*	50%	.5	2/3	2/3	2/3				2.6				
*	50%	.5	1/2	1/2	1/2				3.6				
*	25%	.5	2/3	2/3	2/3				2.7				
*	25%	.5	1/2	1/2	1/2				3.6				
*	75%	2	2/3	2/3	2/3				3.4		3.2	3.4	3.3
*	75%	2	1/2	1/2	1/2				4.1				
*	50%	2	2/3	2/3	2/3				3.5				
*	50%	2	1/2	1/2	1/2				4.2				
*	25%	2	2/3	2/3	2/3				3.6		3.4	3.5	3.5
*	25%	2	1/2	1/2	1/2				4.2				
*	25%	.2	1	1/2	1/2			3.5					
*	50%	.2	1	1/2	1/2			2.8		.63			
*	75%	.2	1	1/2	1/2			2.0		1.3			
*	25%	.5	1	1/2	1/2			4.0					
*	50%	.5	1	1/2	1/2			3.4					
*	75%	.5	1	1/2	1/2			2.5					
*	25%	1	1	1/2	1/2			4.4					
*	50%	1	1	1/2	1/2			3.7		1.9			
*	75%	1	1	1/2	1/2			3.2		2.7			
*	25%	2	1	1/2	1/2			4.9					
*	50%	2	1	1/2	1/2			4.6					
*	75%	2	1	1/2	1/2			4.3					
@	50%	.2	1	1/2	1/2			2.1		.40			
@	75%	.2	1	1/2	1/2			1.3		.59			
@	50%	1	1	1/2	1/2			2.7		1.4			
@	75%	1	1	1/2	1/2			2.2		1.8			
#	50%	.2	1	1/2	1/2			1.0		.32			
#	75%	.2	1	1/2	1/2			.71		.34			
#	50%	1	1	1/2	1/2			1.8		1.2			
#	75%	1	1	1/2	1/2			1.5		1.2			

* Multiple programming levels at the three site are 10/11/11.
@ Multiple programming levels at the three site are 16/8/8.
Multiple programming levels at the three site are 24/4/4.
TZ : Average no. of requests per transaction (transaction size).
DZ : Total number of data items in the database (database size).
MP : Multiprogramming level.
R/(R+W) : Percentage of transactions that are read-only.
IO/Comm : Ratio of local delay to communication delay
(excluding queueing delay).
No. of Copy : Fraction of the database residing at sites S1, S2, & S3.

Figure A.3 Average Response Per Read Request
Short Transactions & Communication Bound

TZ=4, DZ=8192

MP	R/(R +W)	IO / Comm	no. of S1	S2	copy S3	Basic Basic	Prmry Prmry	Cntrl Total	Basic Prmry	Basic Cntrl	Basic Tstmp	Mltpl Versn	Basic Optms
*	25%	.2	1	1	1	12	5.2	4.4	5.3		.2	.28	.29
*	50%	.2	1	1	1	12	5.1	3.5					
*	75%	.2	1	1	1	11	4.9	2.3	4.6		.2	.26	.26
*	75%	1	1	1	1	8.6	4.8	3.5					
*	75%	.5	1	1	1	10	4.8	2.7	3.9				
*	50%	.5	1	1	1				4.9				
*	25%	.5	1	1	1				5.2				
*	75%	2	1	1	1				4.5		2.0	2.2	2.1
*	50%	2	1	1	1				4.9				
*	25%	2	1	1	1				5.3		2.0	2.2	2.1
*	75%	.2	2/3	2/3	2/3	8.3	4.8	6.3	4.9		2.2	2.7	2.6
@	75%	.2	2/3	2/3	2/3			5.2					
#	75%	.2	2/3	2/3	2/3			4.5					
*	75%	.2	1/2	1/2	1/2	6.2	4.7	7.4					
*	50%	.2	2/3	2/3	2/3	8.3	4.9	6.9					
*	50%	.2	1/2	1/2	1/2	6.5	4.7	7.6					
*	25%	.2	2/3	2/3	2/3	8.4	5.0		5.0		2.7	2.9	2.9
*	25%	.2	1/2	1/2	1/2	6.5	4.8						
*	75%	.5	2/3	2/3	2/3			6.3	4.7				
*	75%	.5	1/2	1/2	1/2			7.5	4.8				
*	75%	1	2/3	2/3	2/3			6.5					
*	75%	1	1/2	1/2	1/2			7.5					
*	50%	.5	2/3	2/3	2/3				4.9				
*	50%	.5	1/2	1/2	1/2				4.8				
*	25%	.5	2/3	2/3	2/3				5.0				
*	25%	.5	1/2	1/2	1/2				4.9				
*	75%	2	2/3	2/3	2/3				4.9		3.2	3.6	3.5
*	75%	2	1/2	1/2	1/2				4.9				
*	50%	2	2/3	2/3	2/3				5.1				
*	50%	2	1/2	1/2	1/2				5.1				
*	25%	2	2/3	2/3	2/3				5.2		3.4	3.7	3.6
*	25%	2	1/2	1/2	1/2				5.2				
*	25%	.2	1	1/2	1/2			4.0					
*	50%	.2	1	1/2	1/2			3.2		4.1			
*	75%	.2	1	1/2	1/2			2.1		1.5			
*	25%	.5	1	1/2	1/2			4.1					
*	50%	.5	1	1/2	1/2			3.4					
*	75%	.5	1	1/2	1/2			2.7					
*	25%	1	1	1/2	1/2			4.6					
*	50%	1	1	1/2	1/2			3.9		3.8			
*	75%	1	1	1/2	1/2			3.4		2.1			
*	25%	2	1	1/2	1/2			5.1					
*	50%	2	1	1/2	1/2			4.7					
*	75%	2	1	1/2	1/2			4.4					
@	50%	.2	1	1/2	1/2			2.1		2.9			
@	75%	.2	1	1/2	1/2			1.3		2.2			
@	50%	1	1	1/2	1/2			2.8		3.2			
@	75%	1	1	1/2	1/2			2.4		2.0			
#	50%	.2	1	1/2	1/2			1.2		1.7			
#	75%	.2	1	1/2	1/2			.81		1.6			
#	50%	1	1	1/2	1/2			1.9		2.3			
#	75%	1	1	1/2	1/2			1.6		1.7			

* Multiple programming levels at the three site are 10/11/11.
@ Multiple programming levels at the three site are 16/8/8.
Multiple programming levels at the three site are 24/4/4.
TZ : Average no. of requests per transaction (transaction size).
DZ : Total number of data items in the database (database size).
MP : Multiprogramming level.
R/(R+W) : Percentage of transactions that are read-only.
IO/Comm : Ratio of local delay to communication delay (excluding queueing delay).
No. of Copy : Fraction of the database residing at sites S1, S2, & S3.

Figure A.4 Average Response Per Write Request, Short Transactions & Communication Bound

TZ= 4, DZ= 8192

MP	R/(R +W)	IO / Comm	no. of copy S1	S2	S3	Cntrl Total	Basic Prmry
32	75%	.4/1/2	1	1/2	1/2	6.8/2.3	8.9/2.9
32	25%	.4/1/2	1	1/2	1/2	.93/2.7	.95/2.8
32	75%	2/1/2	1	1/2	1/2	5.6/1.9	6.5/2.1
32	25%	2/1 2	1	1/2	1/2	.91/2.7	.98/2.9
32	75%	.4/1/8	1	1/2	1/2	3.3/1.1	3.8/1.2
32	25%	.4/1/8	1	1/2	1/2	.39/1.2	.41/1.2
32	75%	2/1/8	1	1/2	1/2	3.1/1.1	3.6/1.2
32	25%	2/1/8	1	1/2	1/2	.40/1.2	
32	75%	.4/1/2	2/3	2/3	2/3	2.6/.87	
32	25%	.4/1/2	2/3	2/3	2/3	.66/1.9	
32	75%	2/1/2	2/3	2/3	2/3	2.6/.85	
32	25%	2/1 2	2/3	2/3	2/3	.75/1.9	
32	75%	.4/1/8	2/3	2/3	2/3		
32	25%	.4/1/8	2/3	2/3	2/3		
32	75%	2/1/8	2/3	2/3	2/3		
32	25%	2/1/8	2/3	2/3	2/3		
32	75%	.4/1/2	1	1/2	1/2	2.9/3.1	1.9/3.5
32	25%	.4/1/2	1	1/2	1/2	6.7/6.3	1.1/7.3
32	75%	2/1/2	1	1/2	1/2	4.0/4.1	3.5/3.6
32	25%	2/1 2	1	1/2	1/2	6.9/7.0	2.4/7.6
32	75%	.4/1/8	1	1/2	1/2	5.9/5.3	4.3/16
32	25%	.4/1/8	1	1/2	1/2	16/14	5.5/6.9
32	75%	2/1/8	1	1/2	1/2	6.8/6.7	
32	25%	2/1/8	1	1/2	1/2	14/14	
32	75%	.4/1/2	2/3	2/3	2/3	10/9	
32	25%	.4/1/2	2/3	2/3	2/3	11/12	
32	75%	2/1/2	2/3	2/3	2/3	10/9	
32	25%	2/1 2	2/3	2/3	2/3	11/12	
32	75%	.4/1/8	2/3	2/3	2/3		
32	25%	.4/1/8	2/3	2/3	2/3		
32	75%	2/1/8	2/3	2/3	2/3		
32	25%	2/1/8	2/3	2/3	2/3		

TZ = Average number of requests per transaction (transaction size).
DZ = Total number of data items in the database (database size).
MP = Multiprogramming level.
R/(R+W) = Percentage of transactions that are read-only.
IO/Comm = local delay/message communication delay/data communication delay
no. of copy = Fraction of the database residing at each site.

Figure A.5 Through-Put (Read/Write): Short Transactions & Communication Bound

TZ=4, MP=32, DZ=8192.

MP	R/W	IO/Com	Database Copies			Basic Basic	Prmry Prmry	Cntrl	Basic Prmry	Basic Cntrl	Basic Tstmp	Mltpl Versn	Basic Optms
●	.25	.2	1	1	1	.97/2.9	1.4/4.3	1.3/4.0	1.5/4.5		1.8/5.4	1.9/5.6	1.8/5.6
●	.50	.2	1	1	1	2.5/2.6	3.2/3.2	2.8/3.0					
●	.75	.2	1	1	1	5.6/1.8	5.5/1.8	5.0/1.6	7.1/2.5		8.6/2.9	8.1/2.7	8.1/2.7
●	.25	.5	1	1	1	.40/1.3	.63/1.8		.65/2.0				
●	.50	.5	1	1	1	1.1/1.1	1.4/1.4		1.6/1.6				
●	.75	.5	1	1	1	2.3/.79	2.4/.80	2.3/.76	3.1/1.0				
●	.25	1	1	1	1	.21/.64	.34/.94		.34/1.0				
●	.50	1	1	1	1	.58/.55	.72/.70		.81/.81				
●	.75	1	1	1	1	1.2/.39	1.2/.39	1.3/.39	1.6/.53				
●	.25	2	1	1	1	.11/.32					.21/.59	.21/.59	.22/.60
●	.50	2	1	1	1	.28/.28							
●	.75	2	1	1	1	.61/.20					.94/.31	.85/.28	.84/.28
●	.25	.2	2/3	2/3	2/3	1.2/3.6	1.5/4.7	1.3/3.9	1.5/4.7		1.7/4.9	2.0/6.0	2.0/5.9
●	.50	.2	2/3	2/3	2/3	2.9/2.9	3.3/3.3	2.9/2.8					
●	.75	.2	2/3	2/3	2/3	5.6/1.8	5.6/1.8	4.7/1.6	6.2/2.1		7.1/2.5	7.8/2.4	7.7/2.6
@	.75	.2	2/3	2/3	2/3			4.1/1.3					
#	.75	.2	2/3	2/3	2/3			3.4/1.1					
●	.25	.5	2/3	2/3	2/3	.53/1.6	.66/2.0		.68/2.0				
●	.50	.5	2/3	2/3	2/3	1.3/1.2	1.5/1.5		1.5/1.5				
●	.75	.5	2/3	2/3	2/3	2.3/.79	2.4/.80	2.2/.72	2.7/.84				
●	.25	1	2/3	2/3	2/3	.26/.81			.34/1.0				
●	.50	1	2/3	2/3	2/3	.63/.64			.80/.77				
●	.75	1	2/3	2/3	2/3	1.2/.39		1.1/.39	1.3/.46				
●	.25	1	2/3	2/3	2/3						.18/.54	.24/.72	.23/.71
●	.75	1	2/3	2/3	2/3						.78/.26	1.0/.32	.88/.29
●	.25	.2	1/2	1/2	1/2	1.4/4.3	1.6/4.9						
●	.50	.2	1/2	1/2	1/2	3.3/3.2	3.6/3.6						
●	.75	.2	1/2	1/2	1/2	5.7/1.9	5.8/1.8	4.6/1.5					
●	.25	.5	1/2	1/2	1/2	.60/1.8	.71/2.1		.66/2.2				
●	.50	.5	1/2	1/2	1/2	1.4/1.4	1.5/1.5		1.6/1.6				
●	.75	.5	1/2	1/2	1/2	2.5/.78	2.4/.88		2.8/.86				
●	.25	1	1/2	1/2	1/2	.32/.94	.37/1.1		.36/1.1				
●	.50	1	1/2	1/2	1/2	.72/.71	.80/.76		.81/.81				
●	.75	1	1/2	1/2	1/2	1.2/.43	1.3/.42		1.4/.47				
●	.25	.2	1	1/2	1/2			1.2/3.7					
●	.50	.2	1	1/2	1/2			2.8/2.7		3.1/3.2			
●	.75	.2	1	1/2	1/2			4.6/1.4		5.8/1.9			
●	.25	.5	1	1/2	1/2			.54/1.7					
●	.50	.5	1	1/2	1/2			1.2/1.2					
●	.75	.5	1	1/2	1/2			2.0/.67					
●	.25	1	1	1/2	1/2			.30/.85					
●	.50	1	1	1/2	1/2			.63/.63		.74/.68			
●	.75	1	1	1/2	1/2			1.1/.34		1.3/.41			
@	.50	.2	1	1/2	1/2			2.4/2.4		2.6/2.7			
@	.75	.2	1	1/2	1/2			4.0/1.3		5.2/1.6			
@	.50	1	1	1/2	1/2			.54/.54		.62/.60			
@	.75	1	1	1/2	1/2			.90/.31		1.1/.38			
#	.50	.2	1	1/2	1/2			2.0/2.1		2.2/2.2			
#	.75	.2	1	1/2	1/2			3.4/1.1		4.0/1.3			
#	.50	1	1	1/2	1/2			.45/.44		.48/.49			
#	.75	1	1	1/2	1/2			.75/.25		.88/.28			

● Multiple programming levels at the three site are 10/11/11.
@ Multiple programming levels at the three site are 16/8/8.
Multiple programming levels at the three site are 24/4/4.

Assumptions:
Queueing for local processing is simulated.
Two kinds of local processing delay are simulated:
message processing delay and data processing delay.
The average round trip communication is fixed at 1
The message processing delay is fixed at 5% of the
5% of round trip communication delay
Ratio of data processing & message processing delay is 10
The ratio of data processing delay to round trip
communication delay is shown in column 'IO/Com'

Notation:
TZ = Average number of requests per transaction (transaction size).
DZ = Total number of data items in the database (database size).
MP = Multiplr programming level.
R/W = Percentage of transactins that are read-only.
IO/Com = Ratio of local data processing delay to communication
delay (excluding queueing).
Database Copies = Fraction of the database residing at each site.

Figure A.6 Through-Put (Read/Write), Short Transactions & IO Bounded

TZ=4, MP= 32, DZ= 8192

MP	R/W	IO/Com	Database Copies			Basic Basic	Prmry Prmry	Cntrl Total	Basic Prmry	Basic Cntrl	Basic Tstmp	Mltpl Versn	Basic Optms
*	.25	.2	1	1	1	3.6/8.7	4.0/5.4	5.5/5.7	2.3/5.5		3.0/3.0	3.4/3.5	3.4/3.5
*	.50	.2	1	1	1	3.3/8.1	4.0/5.4	5.3/5.4					
*	.75	.2	1	1	1	2.9/7.3	4.0/5.3	5.0/5.1	2.2/5.6		2.3/2.3	2.9/3.0	2.9/3.1
*	.25	.5	1	1	1	9.0/20	9.2/12		5.5/12				
*	.50	.5	1	1	1	8.0/18	9.1/12		5.4/12				
*	.75	.5	1	1	1	7.1/16	9.0/12	11/11	5.4/13				
*	.25	1	1	1	1	17/38	18/24		11/24				
*	.50	1	1	1	1	16/35	18/24		11/24				
*	.75	1	1	1	1	14/31	18/24	20/20	11/24				
*	.25	2	1	1	1	35/76					27/27	33/33	33/33
*	.50	2	1	1	1	32/70							
*	.75	2	1	1	1	28/61					22/22	29/29	29/29
*	.25	.2	2/3	2/3	2/3	3.8/6.6	4.6/4.7	5.8/6.0	3.2/4.9		3.7/3.6	2.9/3.3	3.4/3.5
*	.50	.2	2/3	2/3	2/3	3.7/6.4	4.4/4.6	5.5/5.7					
*	.75	.2	2/3	2/3	2/3	3.5/6.2	4.3/4.4	5.4/5.5	[illegible].2/5.0		3.0/3.0	2.4/2.7	3.2/3.4
@	.75	.2	2/3	2/3	2/3			6.1/6.2					
#	.75	.2	2/3	2/3	2/3			7.1/7.3					
*	.25	.5	2/3	2/3	2/3	9.2/15	11/11		7.4/11				
*	.50	.5	2/3	2/3	2/3	8.8/15	10/10		7.5/11				
*	.75	.5	2/3	2/3	2/3	8.4/14	9.8/9.9	11/12	7.6/11				
*	.25	1	2/3	2/3	2/3	18/29			15/22				
*	.50	1	2/3	2/3	2/3	17/28			15/22				
*	.75	1	2/3	2/3	2/3	16/27		22/22	15/22				
*	.25	1	2/3	2/3	2/3						33/34	26/28	30/30
*	.75	1	2/3	2/3	2/3						12/12	20/21	29/29
*	.25	.2	1/2	1/2	1/2	4.0/5.4	4.2/4.5						
*	.50	.2	1/2	1/2	1/2	3.8/5.2	4.2/4.4						
*	.75	.2	1/2	1/2	1/2	3.7/5.1	4.1/4.3	5.5/5.7					
*	.25	.5	1/2	1/2	1/2	9.2/12	9.9/10		7.4/10				
*	.50	.5	1/2	1/2	1/2	9.1/12	9.7/9.9		7.5/10				
*	.75	.5	1/2	1/2	1/2	8.8/11	9.4/9.6		7.6/11				
*	.25	1	1/2	1/2	1/2	18/24	19/20		15/20				
*	.50	1	1/2	1/2	1/2	18/23	19/19		15/20				
*	.75	1	1/2	1/2	1/2	17/22	18/19		15/21				
*	.25	.2	1	1/2	1/2			6.0/6.1					
*	.50	.2	1	1/2	1/2			5.7/5.9		3.1/6.4			
*	.75	.2	1	1/2	1/2			5.5/5.6		3.2/6.7			
*	.25	.5	1	1/2	1/2			13/14					
*	.50	.5	1	1/2	1/2			13/13					
*	.75	.5	1	1/2	1/2			12/12					
*	.25	1	1	1/2	1/2			26/26					
*	.50	1	1	1/2	1/2			25/25		14/29			
*	.75	1	1	1/2	1/2			23/24		14/30			
@	.50	.2	1	1/2	1/2			6.3/6.4		4.0/6.9			
@	.75	.2	1	1/2	1/2			6.0/6.2		3.7/7.0			
@	.50	1	1	1/2	1/2			28/28		17/31			
@	.75	1	1	1/2	1/2			27/27		17/32			
#	.50	.2	1	1/2	1/2			7.0/7.2		5.4/7.4			
#	.75	.2	1	1/2	1/2			6.7/6.8		5.0/7.3			
#	.50	1	1	1/2	1/2			32/33		25/34			
#	.75	1	1	1/2	1/2			31/31		23/34			

* Multiple programming levels at the three site are 10/11/11.
@ Multiple programming levels at the three site are 16/8/8.
# Multiple programming levels at the three site are 24/4/4.

Assumptions:
Queueing for local processing is simulated.
Two kinds of local processing delay are simulated:
message processing delay and data processing delay.
The average round trip communication is fixed at 1
The message processing delay is fixed at 5% of the
5% of round trip communication delay
Ratio of data processing & message processing delay is 10
The ratio of data processing delay to round trip
communication delay is shown in column 'IO/Com'

Notation:
TZ = Average number of requests per transaction (transaction size).
DZ = Total number of data items in the database (database size).
MP = Multiplr programming level.
R/W = Percentage of transactins that are read-only.
IO/Com = Ratio of local data processing delay to communication
delay (excluding queueing).
Database Copies = Fraction of the database residing at each site.

Figure A.7 Average Response Time (Read/Write):
Short Transactions & IO Bound

TZ=16, DZ=8192, MP=32

MP	R/W	IO/Com	Database Copies			Basic Prmry	Basic Tstmp	Mltpl Versn	Basic Optms
*	.25	.2	1	1	1	2.0/6.0	1.5/4.4	.90/.20	.85/1.9
*	.75	.2	1	1	1	9.2/3.0	7.9/2.8	6.6/2.1	6.6/2.4
*	.25	2	1	1	1	.25/.83	.16/.46	.09/.20	.09/.20
*	.75	2	1	1	1	1.1/.37	.90/.28	.69/.22	.95/.29
*	.25	.2	2/3	2/3	2/3	1.8/5.6	1.4/3.9	2.1/6.1	2.1/6.2
*	.75	.2	2/3	2/3	2/3	7.9/2.7	6.7/1.9	10./3.5	9.6/3.4
*	.25	2	2/3	2/3	2/3	.25/.81	.14/.39	.26/.78	.23/.85
*	.75	2	2/3	2/3	2/3	.94/.33	.68/.22	1.4/.44	1.3/.37

* Multiple programming levels at the three site are 10/11/11.
Ratio of local data processing & message processing delay is 10

Assumption:
Queueing for local processing is simulated.
Two kinds of local processing are simulated:
(message and data processing).
The round trip communication is fixed at 1
The local message processing delay is fixed at
5% of the round trip communication delay
The ratio of local data processing delay to round trip
communication delay is shown in colume 'IO/Comm'

Notation:
TZ = Average number of requests per transaction.
DZ = Total number of data items in the database.
MP = Multiple programming level.
R/W = Ratio of read-only to write transactions.
IO/Com = Ratio of local data processing delay to
communication delay (excluding queueing).
Database Copies = Fraction of the database at each site.

Figure A.8 Through-Put (Read/Write): Long Transaction Loaded & IO Bound

TZ=16,DZ=8192,MP=32

MP	R/W	IO/ Com	Database Copies			Basic Prmry	Basic Tstmp	Mltpl Versn	Basic Optms
*	.25	.2	1	1	1	2.8/4.6	2.2/2.2	1.1/2.7	2.1/2.6
*	.75	.2	1	1	1	1.9/5.5	2.2/2.2	1.6/2.8	1.7/3.0
*	.25	2	1	1	1	20./33	22./22	13/25	19/24
*	.75	2	1	1	1	17./42	21/21	19/26	21/27
*	.25	.2	2/3	2/3	2/3	3.3/4.8	3.0/3.0	1.4/2.6	2.3/2.9
*	.75	.2	2/3	2/3	2/3	2.4/5.6	2.9/2.9	1.9/2.6	2.0/3.4
*	.25	1	2/3	2/3	2/3	25./33	29/29	14/19	18/21
*	.75	1	2/3	2/3	2/3	22./42	28/28	16/20	18/25

* Multiple programming levels at the three site are 10/11/11.
Ratio of local data processing & message processing delay is 10

Assumption:
Queueing for local processing is simulated.
Two kinds of local processing are simulated:
(message and data processing).
The round trip communication is fixed at 1
The local message processing delay is fixed at
5% of the round trip communication delay
The ratio of local data processing delay to round trip
communication delay is shown in colume "IO/Comm"

Notation:
TZ = Average number of requests per transaction.
DZ = Total number of data items in the database.
MP = Multiple programming level.
R/W = Ratio of read-only to write transactions.
IO/Com = Ratio of local data processing delay to
communication delay (excluding queueing).
Database Copies = Fraction of the database at each site.

Figure A.9 Average Response Time: Long
Transaction Loaded & IO Bound

TZ=16,DZ=8192,MP=32

MP	R/W	IO/Com	Database Copies			Basic Prmry	Basic Tstmp	Mltpl Versn	Basic Optms
*	.25	.2	1	1	1	2.4/7.3	9/26	3.6/8.6	4.3/10
*	.75	.2	1	1	1	21 /6.5	64/19	38/11	40/12
*	.25	2	1	1	1	1.2/3.5	1.0/2.8	.42/.98	.46/1.3
*	.75	2	1	1	1	5.4/1.8	8.0/2.2	10/2.6	6.2/1.9
*	.25	.2	2/3	2/3	2/3	2.2/6.5	1.6/4.4	.97/2.7	1.9/5.1
*	.75	.2	2/3	2/3	2/3	9.9/3.2	7.9/2.4	7.5/2.8	10/3
*	.25	2	2/3	2/3	2/3	1.0/2.9	.84/2.4	.54/1.5	.66/2.0
*	.75	2	2/3	2/3	2/3	4.7/1.5	4.0/1.3	5.6/1.7	4.8/1.3

* Multiple programming levels at the three site are 10/11/11.

Assumption:
Queueing for communication channel is simulated.
Only one kind of local processing is simulated.
The average round trip communication is fixed at 1
The ratio of local data processing delay to round trip
communication delay is shown in colume "IO/Comm"

Notation:
TZ = Average number of requests per transaction.
DZ = Total number of data items in the database.
MP = Multiple programming level.
R/W = Ratio of read-only to write transactions.
IO/Com = Ratio of local processing delay to communication
delay (excluding queueing delay).
Database Copies = Fraction of the database at each site.

Figure A.10 Through-Put (Read/Write): Long Transaction
Loaded & Communicaton Bound

MP	R/W	IO/Com	Database Copies			Basic Prmry	Basic Tstmp	Mltpl Versn	Basic Optms
*	.25	.2	1	1	1	1.2/4.1	.2/.2	.2/.53	.42/.51
*	.75	.2	1	1	1	.52/3.1	.2/.2	.2/.45	.30/.50
*	.25	2	1	1	1	3.9/7.8	2/2	2/4.9	3.5/4.6
*	.75	2	1	1	1	3.1/8.8	2/2	2/3.1	2.8/4.3
*	.25	.2	2/3	2/3	2/3	2.5/4.2	2/2	.86/3.7	2.2/2.8
*	.75	.2	2/3	2/3	2/3	2.1/4.2	1.9/1.9	1.4/3.3	1.9/3.1
*	.25	1	2/3	2/3	2/3	6.3/8.9	3.1/3.2	3.0/7.8	5.6/6.7
*	.75	1	2/3	2/3	2/3	4.2/8.5	3.2/3.1	3.1/5.7	4.2/6.6

* Multiple programming levels at the three site are 10/11/11.
Assumption:
Queueing for communication channel is simulated.
Only one kind of local processing is simulated.
The average round trip communication is fixed at 1
The ratio of local data processing delay to round trip
communication delay is shown in column "IO/Comm"

Notation:
TZ = Average number of requests per transaction.
DZ = Total number of data items in the database.
MP = Multiple programming level.
R/W = Ratio of read-only to write transactions.
IO/Com = Ratio of local processing delay to communication
delay (excluding queueing delay).
Database Copies = Fraction of the database at each site.

Figure A.11 Average Response Time (Read/Write)
Long Transaction & Communication Bound